Universidade de Aveiro
2014
Departamento de Eletrónica, Telecomunicações e Informática

Daniela Rodrigues Valério

Componentes Gráficos em HTML5

Universidade de Aveiro
2014
Departamento de Eletrónica, Telecomunicações e Informática

Daniela Rodrigues Valério

Componentes Gráficos em HTML5

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Professor Doutor José Luís Guimarães Oliveira, Professor Associado do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro.

O trabalho descrito nesta dissertação foi realizado na empresa Nokia Networks, sob a supervisão do Eng. Nuno Miguel Sousa, Engenheiro e do Engenheiro Nuno Miguel Sousa, Engenheiro de Desenvolvimento de Software da empresa Nokia Networks.

Dedico este trabalho aos meus pais, pelo incansável apoio que sempre me dedicaram.

o júri

Presidente	Professor Doutor Joaquim João Estrela Ribeiro Silvestre Madeira Professor Auxiliar, Universidade de Aveiro
Vogal - Arguente Principal	Professor Doutor Rui Pedro Sanches de Castro Lopes Professor Coordenador, Instituto Politécnico de Bragança
Vogal - Orientador	Professor Doutor José Luís Guimarães Oliveira Professor Associado, Universidade de Aveiro

agradecimentos

Agradeço a todos que me deram apoio e esclarecimentos para que este trabalho seja hoje uma realidade, principalmente aos meus professores pelo conhecimento transmitido ao longo dos anos.

palavras-chave

HTML5, JavaScript, AngularJS, Gráficos, Visualização de informação, Tabelas, TDD, CSS, JSON, MVC, Componentes Web

resumo

Este trabalho visa a criação de vários componentes gráficos em HTML5, CSS e JavaScript. Os componentes gráficos implementados permitem várias formas de visualização de dados e manipulação dos mesmos, desde diferentes tipos de tabelas a diferentes tipos de gráficos. Com diferentes níveis de complexidade. Estes componentes possibilitam um elevado nível de interação entre os dados e o utilizador. Os dados a apresentar são recebidos em formato JSON através de pedidos HTTP e são posteriormente processados para integração com os componentes.

A arquitetura modular e escalável deste projeto permite que os componentes web implementados sejam facilmente reutilizáveis. Podendo, portanto, serem integrados em diferentes produtos, plataformas e outras tecnologias web.

Para a concretização deste projeto recorreu-se a várias linguagens de programação, bem como a várias ferramentas web da atualidade.

keywords

HTML5, JavaScript, AngularJS, Charts, Visualization of information, Tables, Test Driven Development, CSS, JSON, Model View Controller, Web Components

abstract

This work aims at creating an application comprising several types of graphical components in HTML5, CSS and JavaScript. Several kinds of components were implemented for visualization and manipulation of data, from different types of tables to different types of charts. They have several levels of complexity. The implemented components offer a high level of interaction between the user and the data. The data are received in JSON format via HTTP requests, which are later processed by the application.

This project's modular and scalable architecture allows reusing the implemented web components. They can, therefore, be integrated in different products, platforms and other web technologies.

To implement this project a variety of programming languages were used, as well as various web tools.

Conteúdo

Índice de Figuras

Índice de Tabelas

Lista de Acrónimos

AJAX Asynchronous JavaScript and XML
API Application Programming Interface
ASP Active Server Pages
CORS Cross-Origin Resource Sharing
CRUD Create Read Update and Delete
CSS Cascading Style Sheet
CVS Concurrent Versions System
DOM Document Object Model
DHTML Dynamic HTML
ECMA European Computer Manufacturers Association
E2E End-to-End
GUI Graphical User Interface
HTML HyperText Markup Language
HTTP Hypertext Transfer Protocol HyperText Markup Language
IDE Integrated Development Environment
IE Internet Explorer
ISO International Organization for Standardization
JS JavaScript
MIT Massachusetts Institute of Technology
MVVM Model-View-View Model
MVC Model-View-Controller
MVP Model-View-Presenter
MV* ou MVW Model-View-Whatever
NPM Nokia Performance Manager
NSN Nokia Solutions and Networks
OS Operating System
OOP Object-Oriented Programming
PC Personal Computer
POC Proof of Concept
REST Representational State Transfer
RIA Rich Internet Application
SPA Single-page application
SVG Scalable Vector Graphics
TDD Test-Driven Development
UI User Interface
URL Uniform Resource Locator

UX User Experience

W3C World Wide Web Consortium

WHATWG Web Hypertext Application Technology Working Group

XML Extensible Markup Language

YUI Yahoo! User Interface

1 Introdução

1.1 Motivação

Nos últimos anos, a área de implementação de aplicações web tem sofrido um desenvolvimento surpreendente. Existe uma constante necessidade de inovar e de responder às novas necessidades dos utilizadores. Todos os dias surgem novas aplicações web, novas ferramentas, novos dispositivos eletrónicos com elevado desempenho e com capacidades de ligação web. Portanto, as empresas têm de estar atentas e preparadas para este desenvolvimento tecnológico e tentar acompanhá-lo.

O projeto da dissertação enquadra-se com a empresa Nokia Networks, anteriormente denominada Nokia Solutions and Networks, (NSN), que tem sentido nos últimos tempos, esta mesma necessidade, de atualizar as ferramentas e linguagens utilizadas nos seus produtos. De momento, um dos projetos da Nokia Networks tem um produto de grande dimensão que, por questões de desatualização das suas tecnologias, tem sentido grandes dificuldades em implementar as novas funcionalidades pedidas pelos seus clientes. O produto em causa está implementado em Java e Flex, sendo o Flex responsável pelo interface do utilizador. Neste sentido, é o código Flex que precisa rapidamente de ser substituído por uma linguagem web atual. Contudo, a quantidade de código Flex que precisa ser migrada é enorme, e por isso, a migração de código será implementada de forma faseada, sendo a primeira fase a substituição da apresentação e manipulação de dados que é feita, ou seja, as tabelas e os vários tipos de gráficos do produto. Com base nesta necessidade, surge a proposta de dissertação deste projeto.

Tendo em conta que a informação só é útil se fizer sentido e esta só fará sentido se manipulada de forma correta, surge a necessidade de implementar ferramentas interativas capazes de auxiliar o utilizador a manipular conjuntos dados, sejam eles de grandes ou de pequenas dimensões. Com a possibilidade de migração do produto, surge também a possibilidade de inovar. As potencialidades e eficácia das novas linguagens e ferramentas web têm de ser aproveitadas para se conseguir uma aplicação web apelativa e interativa.

Nos últimos anos têm surgido cada vez mais ferramentas e tecnologias de apoio ao desenvolvimento de aplicações web. Os clientes estão cientes destas inovações e são cada vez mais exigentes, estando cada vez mais interessados em poder manipular de forma personalizada a aplicação. Tendo em conta que a escolha da tecnologia pode ser decisiva para o sucesso ou insucesso de um projeto, estas tem de ser bem ponderadas e analisadas. Principalmente tendo em conta o produto onde os componentes web serão integrados. É esperado que o produto tenha

de ser mantido por um longo período de tempo e por consequência, que os componentes precisem de o mínimo de esforço possível de manutenção.

1.2 Objetivos

A proposta de dissertação tem como objetivo implementar componentes web para visualização de dados e sua manipulação. É esperado um elevado nível de interação entre componentes e utilizador de forma a torná-los inovadores, mas tendo sempre em conta a necessidade de implementar componentes fáceis de compreender e utilizar. Os componentes a implementar têm de ser modulares, reutilizáveis e o mais independente possíveis. Estes têm de permitir ao utilizador escolher diferentes formas de visualização de informação, que podem ser tabelas ou diferentes tipos de gráficos. Os gráficos podem por exemplo ser: de linhas, colunas, áreas, barras, tarte, *spline*, área *spline* e outros tipos de gráficos mais complexos. Os componentes web têm de ser compatíveis com os navegadores web mais recentes e possibilitarem a sua integração em diferentes produtos da empresa Nokia Networks, não sendo o projeto de código aberto fora da empresa.

Quando se fala de visualização e manipulação de dados é extremamente importante que, entre utilizador e aplicação, exista um elevado nível de interação e que esta seja visível em tempo real. As alterações efetuadas pelo utilizador ou por ações da aplicação deverão ser visíveis para ambos o mais rapidamente possível, permitindo assim uma interação harmoniosa.

Como referido na secção da motivação, um dos focos deste projeto é servir de rampa de lançamento para uma futura e inevitável migração de código de produtos já existentes na empresa. Portanto, é necessário escolher as ferramentas e linguagens que permitam uma integração em partes, sem comprometer o funcionamento das restantes partes implementadas em Flex. A integração dos componentes web não pode comprometer o normal funcionamento do produto.

Os componentes web permitem a implementação dos nossos próprios elementos de HTML5 *(HyperText Markup Language),* encapsulando toda a complexidade e lógica implementada. Desta forma, qualquer programador, mesmo sem grandes conhecimentos em programação *front-end,* pode facilmente implementar uma aplicação utilizando componentes web. Deste modo, da mesma forma que existem os elementos HTML5 div, table, entre outros, implementando componentes web é possível definir os nossos elementos HTML5, como por exemplo nokiaAreaChart, nokiaTable, ou com qualquer outro nome que se pretenda.

Por fim, o projeto desenvolvido tem de ser escalável e extensível para que implementações de novos componentes web possam ser facilmente integrados entre si e no produto. No fundo,

pretende-se implementar uma ferramenta interna para a Nokia, composta por um conjunto de componentes web HTML5, que facilite posteriormente a implementação de funcionalidades do *front-end.*

1.3 Estrutura da Dissertação

Na dissertação são inicialmente analisadas as linguagens de programação web, dando mais enfase às mais usadas no desenvolvimento de aplicações web da atualidade. Posteriormente são analisadas as ferramentas web mais populares e com melhor feedback na comunidade de programadores web. No fim de cada análise de tecnologia é feita uma pequena conclusão com as razões com as quais se decidiu escolher ou não essa mesma tecnologia.

A dissertação está dividida em cinco capítulos. No próximo capítulo, capítulo 2, será feita uma análise às principais linguagens web: HTML, CSS *(Cascading Style Sheet),* Flex, Dart e JavaScript. Neste mesmo capítulo, também serão abordadas as ferramentas de JavaScript mais utilizadas: jQuery, ExtJS, AngularJS e BackboneJS, bem como as APIs *(Application Programming Interfaces)* de JavaScript mais importantes na implementação de gráficos: HighchartsJS, D3 e Google Chart Tools.

No capítulo 3 será apresentada a arquitetura, os requisitos do projeto e as ferramentas de apoio no desenvolvimento. Será também apresentada a estrutura de diretórios e serão abordados os conceitos usados na implementação da aplicação web.

No capítulo 4 será feita uma descrição detalhada da implementação. Neste capítulo são apresentados os componentes web desenvolvidos, as suas diretivas, bem como os serviços e controladores que tiveram de ser implementados. Posteriormente neste capítulo serão também abordados os testes de unidade e testes *end-to-end* que tipicamente acompanham a metodologia ágil TDD (*Test-Driven Development).*

Por fim, no último capítulo, serão apresentadas as conclusões e apontadas as melhorias ao projeto que poderão ser implementadas no futuro.

2 Tecnologias Web

2.1 Introdução

Com a melhoria da qualidade das ligações de rede e o aumento do desempenho dos interpretadores de JavaScript nos navegadores, surgem cada vez mais aplicações web com elevados padrões de qualidade. A experiência entre o utilizador e as aplicações tem sido cada vez mais positiva. Uma boa aplicação tem de ter como principal objetivo responder às expectativas e necessidades dos utilizadores.

A representação de informação tem um grande impacto no utilizador e por isso é preciso prestar extrema importância de como é feita. Pode, à primeira vista, parecer um conceito simples de implementar, mas tornar uma aplicação apelativa e com boa usabilidade não é uma tarefa fácil. Provavelmente é por este motivo que existem tantas bibliotecas de apoio no desenvolvimento de aplicações deste género. A representação de informação é uma área muito abrangente e de extrema utilidade. Para fazer a melhor escolha da representação de dados é necessário primeiro perceber a quantidade, o tipo, bem como o nível de interação que é esperado entre o utilizador e esses mesmos dados.

Nos últimos tempos, foram surgindo muitas ferramentas web para dar suporte aos programadores. Contudo, só algumas merecem uma análise mais profunda devido à solidez e evolução positiva que têm sentido. Torna-se muito importante decidir de forma ponderada quais podem e devem ser usadas como mais-valia neste projeto. É neste capítulo que são analisadas as diversas ferramentas web, mas somente se considerou as ferramentas que já possuem um elevado nível de maturidade, pois só assim podemos, com maior certeza, garantir que persistirão no futuro com igual ou maior força. Contudo, existem também várias linguagens de programação web que serão analisadas para concluir qual ou quais melhor se adaptam às necessidades dos dias de hoje e que vão ao encontro dos objetivos pretendidos com este projeto.

2.2 HTML5

O HTML5 é a mais recente versão da popular linguagem web HTML. Esta nova versão está mais direcionada para as novas realidades tecnológicas, contendo muitas funcionalidades específicas para navegadores web móveis. Um exemplo prático desta realidade é facilmente verificado quando, por exemplo, é selecionado um elemento web para introdução de um campo de entrada. Quando é detetado que a aplicação web está a executar num navegador web móvel é

automaticamente lançado no ecrã do dispositivo um teclado virtual, permitindo assim a introdução de dados da parte do utilizador. Esta funcionalidade é lançada de forma automática, sem que o programador se tenha de ter preocupado com a implementação desta funcionalidade. Existem, no entanto, muitas outras funcionalidades e que executam em ambos os navegadores web, quer móveis ou não móveis. Por exemplo, o novo tipo de elemento de entrada, o tipo email, que automaticamente faz a validação do campo introduzido pelo utilizador, verificando se tem a estrutura esperada de um email. Sem dúvida que as novas funcionalidades introduzidas no HTML5 têm, por isso, como principal objetivo, simplificar a implementação de aplicações web da atualidade.

O HTML5 teve o seu primeiro esboço de especificação publicado em 2008. Contudo, por ser uma linguagem tão extensa não existe ainda uma recomendação W3C *(World Wide Web Consortium)* completa. Esta tarefa torna-se complicada por se tratar de uma linguagem em constante crescimento.

O HTML5 vem ao encontro das novas necessidades tecnológicas e permite, de forma simplificada, a implementação das funcionalidades web mais usuais. Por exemplo, reprodução de áudio e vídeo, bem como a criação de jogos e representação de imagens vetoriais.

Das várias funcionalidades do HTML5 foram destacadas e agregadas em sete categorias as mais comuns, as quais estão definidas na Figura 2.1.

Figura 2.1 – Funcionalidades do HTML5

Para melhor compreensão das categorias consideradas é feito uma breve explicação de cada uma:

- MOVIMENTO: Os objetos das páginas web movem-se e reagem a movimentos do cursor.
- TIPOLOGIA: As páginas web podem ser enriquecidas com estilos e formatações de fontes, cores, entre outros.

- ARMAZENAMENTO: Os dados podem ser armazenados localmente num computador ou num dispositivo móvel, de forma a permitir que a aplicação consiga operar mesmo *offline*.

- 3D: Utilizando a tecnologia WebGL é possível criar efeitos interativos 3D usando o processador gráfico do dispositivo onde a aplicação está a executar.

- JOGOS, VÍDEO e ÁUDIO: Os jogos interativos e a reprodução de vídeo e áudio conseguem executar sobre o navegador web, evitando a instalação de *plugins*.

Para além destas funcionalidades, existem outras mais subtis. Por exemplo, a capacidade de validação automática de campos de formulários antes de se submeter esses dados no servidor. Esta validação, como é feita do lado do cliente, permite que não existam pedidos desnecessários, evitando a sobrecarga do servidor.

O HTML5 agrega diversas tecnologias web muito conhecidas da comunidade web. Como por exemplo CSS3, SVG (*Scalable Vector Graphics*), bem como o elemento canvas, que possibilita o desenho 2D. Integra também a API WebSocket que facilita a implementação de ligações na rede entre servidor e navegador do cliente, como também integra a API PostMessage que permite a troca de mensagens entre *scripts* de diferentes domínios. Estas são somente algumas das muitas tecnologias. Como se pode observar na Figura 2.2, o HTML5 abrange diversas tecnologias, algumas ainda sem recomendação W3C.

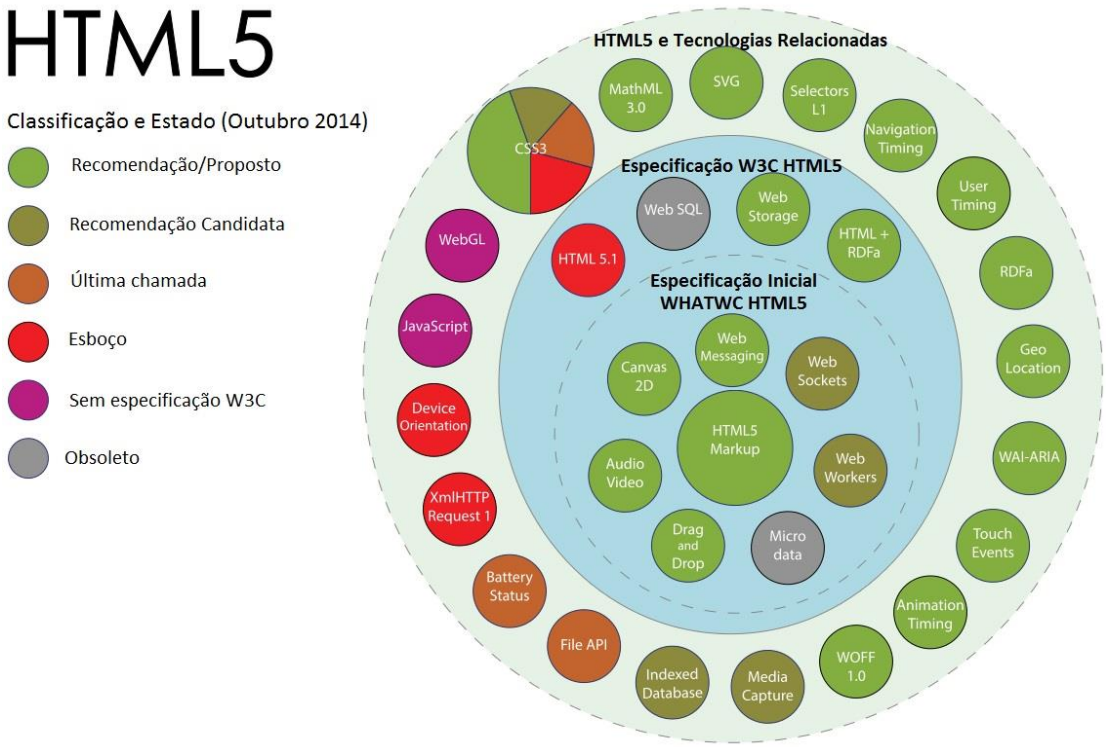

Figura 2.2 – HTML5 e tecnologias relacionadas [1]

Esta linguagem, com o tempo, tem ganho novas ramificações e muitas tecnologias relacionadas.

Um aspeto muito positivo desta linguagem é ser totalmente compatível com antigas aplicações web HTML. O HTML5 permite assim evitar o *refactoring* de código. As aplicações mais antigas continuam assim funcionais sem existir necessidade de o programador se ter de adaptar à versão HTML5. Este fator facilitou a aceitação desta nova versão pela comunidade web. O HTML5 surge não como obrigatoriedade mas como possibilidade de fazer mais e melhor, tornando-se uma linguagem web de escolha óbvia pela maioria dos programadores em detrimento de outras linguagens web.

Por todas estas razões, esta é a linguagem adotada para o desenvolvimento dos *templates* dos componentes web do projeto. Esta linguagem contém muita informação de suporte no desenvolvimento de aplicações web. Sem margem para dúvidas que esta linguagem é a linguagem web do presente e do futuro.

2.3 CSS e CSS3

Cascading Style Sheets, ou como normalmente é conhecido CSS, é uma linguagem de folhas de estilo. Serve para definir regras relativas à apresentação e estilo de um documento escrito em linguagens como HTML ou XML *(Extensible Markup Language)*. O uso de CSS permite uma melhor separação entre o estilo e o conteúdo do documento a formatar.

Com a nova era das tecnologias web, surgiu a versão CSS3, associada também à linguagem HTML5. Esta nova versão dotou o CSS com regras de animações, transições, imagens e outros efeitos interativos. Infelizmente, por ser ainda uma tecnologia muito recente, existem muitas funcionalidades que não são suportadas pela maioria dos navegadores web aos quais os componentes têm de suportar. Por este motivo, nos dias de hoje, o CSS3 não pode ser considerado no âmbito do projeto, tendo-se optado pelo CSS. No entanto, num futuro próximo, o CSS3 será a linguagem de referência no que toca a folhas de estilo.

2.3.1 Orange Touch

O Orange Touch, mais conhecido como OT, é propriedade da empresa Nokia Networks. Trata-se de uma ferramenta que contém as regras, imagens, formatações, cores e todos os estilos que têm de ser adotados em todos os produtos da NSN. Desta forma é possível manter sempre o mesmo *look and feel* nos produtos desenvolvidos pelos programadores da empresa. Os clientes que usam estes produtos já estão familiarizados com estes estilos e formatações. Portanto, era previsível a utilização do Orange Touch para implementar as regras de estilo. Esta biblioteca não é

de código aberto à comunidade web, estando somente disponível para os trabalhadores da Nokia Networks.

O OT é implementado em JavaScript, usando a ferramenta muito conhecida jQuery. Esta ferramenta tem muita documentação de apoio e exemplos que ajudam na compreensão das funcionalidades e potencialidades desta biblioteca de CSS. Infelizmente, esta biblioteca pode ser considerada comente como um POC (*Proof of Concept)* e por este motivo foram necessárias implementar várias correções e mesmo implementação de lógica de negócio para que o que era esperado funciona-se.

Contudo, em Agosto deste ano, a empresa NSN passou a ser a Nokia Networks e portanto, os produtos da NSN têm de, com o tempo, adotar o novo *look and feel* correspondente à Nokia Networks, denominado de Ocean Touch. Com a mudança da empresa NSN para Nokia Networks todas as regras de estilos sofreram alterações. Esta nova ferramenta veio substituir o Orange Touch e apesar de tardiamente à fase de implementação e decisão do projeto, surgiu esta necessidade de mudança que será apresentada num capítulo mais adequado, o capítulo sobre a implementação do projeto.

2.3.2 Bootstrap

O Bootstrap[1] é uma ferramenta com regras de estilo, imagens e entre outras formatações, tal como a ferramenta analisada anteriormente. Esta ferramenta foi e continua a ser bem aceite pela comunidade web. Em pouco tempo, usando esta ferramenta é possível implementar aplicações com um nível visual muito apelativo. Esta permite o redimensionamento automático dos elementos presentes numa página web com base no tamanho detetado de ecrã, entre muitas outras funcionalidades interessantes.

O Bootstrap permite simplificar a vida do programador utilizando, por exemplo, a lógica da divisão da página em grelhas. Desta forma, posicionar os elementos numa página web torna-se simples e intuitivo. Esta lógica de divisão por grelha permite que não sejam precisos fazer ajustes complexos que implicam cálculos de pixéis quando, simplesmente, se pretende reposicionar ou introduzir novos elementos numa página.

Esta ferramenta tem definidas classes que correspondem a regras de estilo de cores, formatos dos elementos da página a apresentar, entre outros. Utilizando essas classes num documento HTML é possível obter automaticamente o estilo correspondente definido nessa classe. Para além das regras de estilo, a ferramenta disponibiliza diferentes tipos de ícones muito intuitivos. As aplicações Bootstrap estão por norma bem estruturadas e são muito intuitivas, tipicamente acompanhadas com animações que realçam o que realmente é importante.

[1] http://getbootstrap.com/

O Boostrap está em franco desenvolvimento. Estão a surgir novos *plugins* que têm tornado esta ferramenta cada vez mais extensa mas que mesmo assim, continua fácil de compreender e utilizar pelo utilizador. De notar, que são disponibilizados muitos exemplos e documentação *online,* bem como muitos tutoriais de ajuda. Como é usada por muitos programadores, existe muito suporte e muitos fóruns para debater ideias e soluções.

O Bootstrap pode e deve ser usado em aplicações em que as regras de estilo não tenham de seguir um estilo rigoroso, sendo por norma, da autoria do programador a decisão final de quais as formatações a adotar.

2.4 Flex

O Flex[2] é uma tecnologia que permite a implementação de aplicações web baseadas na plataforma do Macromédia Flash. Foi lançada em 2004 pela Macromedia e posteriormente adquirida pela Adobe Systems. Mas, em 2011, foi doada pela Adobe para a Apache Software Foundation. Na era dos PCs, a linguagem Flex foi muito importante, mas com a enorme evolução do HTML5 e do JavaScript, o Flex tem vindo a perder força na comunidade web de programadores.

Vivemos numa era de completa submissão às aplicações web e o facto de o Flash não ser suportado, por *default,* pelos sistemas operativos nem móveis, nem *desktop*, tem comprometido fortemente o futuro desta tecnologia. O Flex só pode ser interpretado e executado pelo Adobe Flash Player, em contrapartida, o JavaScript é interpretado por qualquer navegador web por ter integrado o seu próprio interpretador de JavaScript. Este é sem dúvida um grande problema para o futuro do Flex.

Para além do problema associado à necessidade do *plugin* Adobe Flash Player, o Flex não foi desenvolvido para dar apoio às novas necessidade das aplicações web. Falamos de interfaces de toque, animações, reprodução de *media*, que têm de correr tanto em dispositivos *laptop* e *desktop* que disponibilizam bastantes recursos, bem como, em dispositivos móveis de baixo desempenho e recursos reduzidos. Para colmatar estes problemas, foram surgindo, nos últimos tempos, novas alternativas tecnológicas ao Flash e consequentemente ao Flex. Os programadores decidiram utilizar outras tecnologias e acabaram por demonstrar que o Flash já não faz falta e que por tal, também o FLex já não é mais necessário na implementação de aplicações web graficamente ricas e interativas. Todos os dias são implementadas novas aplicações e jogos web que, têm ou tiveram imenso sucesso e que não estão dependentes da necessidade do *plugin* Flash. Este foi sem dúvida um ponto de viragem para o Flash que tem perdido o interesse da

[2] http://www.adobe.com/products/flex.html

comunidade web em geral. Pelo contrário, o HTML5 e JavaScript vão tendo cada vez mais suporte e introdução de novas ferramentas web.

Para garantir assim o futuro de uma aplicação é fortemente recomendado a migração de produtos que usem esta tecnologia. Trata-se de uma questão de tempo até que os utilizadores se cansem de instalar o *plugin* da Adobe Flash ou que o custo para manter e implementar novas funcionalidade na aplicação seja demasiado elevado em comparações com outras tecnologias.

2.5 Dart

O Dart[3] é uma linguagem de programação web que pretende substituir as outras linguagens web da atualidade, mais propriamente, o JavaScript. Esta linguagem pretende ser uma linguagem ainda mais fácil de entender e de programar do que a linguagem JavaScript.

Uma das grandes vantagens do Dart está na sua semelhança com o Java, tornando-a mais fácil de aprender e utilizar, para quem já estiver familiarizado com a lógica e estrutura de programação do Java. Para melhor compreensão do nível de semelhanças inerentes entre estas duas linguagens é apresentado de seguida, um pequeno exemplo de código Dart. Note-se que juntamente ao código foram introduzidos comentários para ajudar na compreensão do mesmo.

```
import ‘dart:math’ show Random;                // Importa a class da biblioteca

void main() {                                  // Início de execução da app
    print(new Die(n: 12).roll());              // Imprime valor de novo objeto
}

class Die {                                    // Define uma classe
    static Random shaker = new Random();       // Define uma variável de classe
    int sides, value;                          // Define variáveis de instância
    String toString() => ‘$value’;             //Define  método  usando  sintaxe
abreviada

    Die({int n: 6}) {                          // Define um construtor
      if (4 <= && n <= 20) {
            sides = n;
      } else {
            throw new ArgumentError(/* */);  //Suporte de erros e exceções
      }
          }
    int roll() {                               // Define um método de instância
      return value = shaker.nextInt(sides) + 1; //Obtém um número aleatório
    }
}
```

As semelhanças entre a linguagem Java e Dart são evidentes. É bastante percetível que o Dart foi desenvolvido com base nas capacidades e características do Java e que adotou muitas

[3] https://www.dartlang.org/

das suas funcionalidades e conceitos de programação. Tal como o Java, o Dart também assenta no conceito de programação orientada a objetos.

Um dos maiores problemas desta linguagem é a necessidade de um compilador Dart-to-JavaScript. Só desta forma a aplicação implementada em Dart é interpretada pelos navegadores web, pois na atualidade só o JavaScript é interpretado, por *default,* pelos principais navegadores web.

Ao contrário das expectativas, a linguagem Dart não está a ser muito bem aceite. Nos primeiros tempos do Dart, a comunidade de programadores web observou que a implementação feita em Dart e posteriormente convertida para JavaScript através do compilador Dart-to-JavaScript resultava num número extremamente elevado de linhas de código em comparação com a mesma implementação feita em JavaScript. Este facto trouxe muito má imagem ao Dart, mas ultimamente, este problema de conversão de Dart para JavaScript tem sido corrigido e cada vez mais a performance de uma aplicação em Dart tem sido superior ao JavaScript. Por outro lado, outro aspeto negativo no Dart é não conseguir ser executado no Internet Explorer 8, o que para os dias de hoje, poderá ainda ser um requisito a suportar.

Sendo esta linguagem bastante imatura e sem possibilidade de se conseguir prever qual o seu futuro, não pode ser considerada para implementar projetos de longo prazo de manutenção. Em conclusão, o Dart não foi utilizado na implementação dos componentes do projeto.

2.6 JavaScript e Ferramentas de Apoio

JavaScript surgiu pela primeira vez em 1995, desenvolvida por Brendan Eich. É uma linguagem de programação interpretada, codificada em scripts e baseada em objetos, com capacidade multiplataforma. Foi inicialmente implementada como parte dos navegadores web Netscape para permitir interação com os utilizadores usando *scripts client-side*. Esta linguagem é leve e de pequenas dimensões, tornando-a fácil de incorporar nos navegadores web e outros tipos de aplicações.

Nos dias de hoje, o JavaScript é a linguagem mais usada em páginas web na programação *client-side.* O Gráfico 2.1 apresenta as percentagens de uso de algumas conhecidas linguagens de programação, onde se pode observar claramente que o JavaScript lidera com grande vantagem.

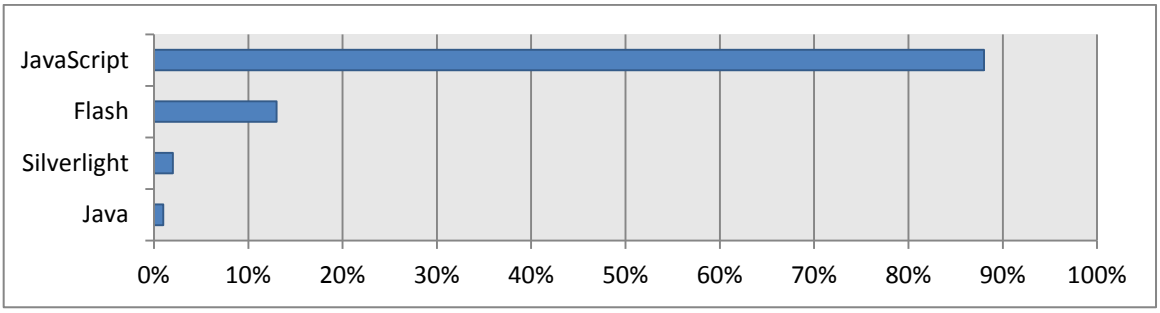

Gráfico 2.1 – Estatísticas das linguagens de programação *cliente-side* [2]

A grande popularidade desta linguagem deve-se principalmente pela sua simplicidade de implementação. O JavaScript assenta num conjunto de paradigmas de programação: scripts, programação orientada a objetos, conceito de objetos *prototype*, programação imperativa e funcional. Apesar da semelhança que esta linguagem tem com as demais linguagens OOP (*Object-Oriented Programming*), como é o caso do Java, o JavaScript não usa o conceito de *class* e a implementação das populares metodologias OOP, como herança, não é feita de forma direta. Como o JavaScript assenta num conceito de objeto protótipo, a herança é dinâmica, quer isto dizer, que o que é herdado pode variar de objeto para objeto individualmente.

Douglas Crockford é muito aclamado e reconhecido pela comunidade de programadores web por ser considerado o grande especialista do JavaScript. Autor do livro muito recomendado: "JavaScript: The Good Parts", Douglas Crockford [3], tem a capacidade de extrair o que de melhor existe no JavaScript, identificando os pontos-chave e as ideias brilhantes que tornam esta linguagem fascinante. Uma das ideias brilhantes desta linguagem está na sua capacidade dinâmica em que os tipos são associados a valores e não a variáveis. Esta característica permite que a mesma variável possa ser inicialmente associada a um número e posteriormente, caso necessário, a uma *string* por exemplo.

Para o programador, esta linguagem permite grande liberdade de desenvolvimento. Este aspeto poderá ser, para alguns, uma grande vantagem mas para outros, uma desvantagem. Esta linguagem distingue-se das demais linguagens clássicas que são muito restritas. Esta linguagem foi definida de tal forma que a falta de ponto e vírgulas é aceitável e que o uso de variáveis e funções globais é um comportamento comum. Esta linguagem não obriga o programador a ter de declarar todas as variáveis e métodos, nem existe o conceito de métodos públicos, privados ou protegidos. Estas características tornam esta linguagem muito simples de usar e compreender. O facto de esta linguagem ser tão flexível e tão expressiva permite que em pouco tempo, se consiga implementar pequenos projetos funcionais.

Com uso do JavaScript a maioria da lógica da interface do usuário passou a ser feita do lado do cliente, *client-side*. Nestes casos o código JavaScript corre localmente do lado do cliente,

ou seja, no navegador web do usuário e por esta razão nota-se uma melhoria significativa do seu desempenho em comparação com aplicações caracterizadas de *server-side*. Nas aplicações *client-side*, estas estão constantemente atentas à deteção de ações do usuário, prontas a despoletar um qualquer evento pretendido, sem obrigatoriamente serem efetuados pedidos para o lado do servidor. Esta capacidade, extremamente útil e poderosa, é impossível de implementar usando somente a linguagem HTML. Este tipo de programação permite comunicação assíncrona e tem controlo do navegador, conseguindo alterar o conteúdo da página que é apresentado. Com o tempo, esta linguagem interpretada, foi-se expandindo para outras vertentes de programação, como é o caso da programação *server-side,* mas sem nunca ter ganho grande domínio nesta área.

O JavaScript é nos dias de hoje, considerada a "linguagem da web". Usando as capacidades desta linguagem é possível efetuar uma infinidade de funcionalidades, sendo também possível efetuar pedidos de informações ao servidor. Os navegadores web têm incorporado interpretadores de JavaScript que permitem, obviamente, interpretar código JavaScript. Com o tempo, os navegadores web têm aumentado o nível de suporte que dão a esta linguagem, tentando também melhorar o desempenho dos seus interpretadores. Com o tempo, vão surgindo extensões para incorporar nas *developer tools* dos diferentes navegadores para ajuda e suporte no desenvolvimento de aplicações web. No caso do Firefox, tem o nome de Firebug. Estes tipos de ferramentas fornecem um conjunto de instrumentos de análise e funcionalidades úteis, como por exemplo, a inspeção de elementos DOM (*Document Object Model*) de uma página HTML, a deteção de erros JavaScript, impressão de informações na consola, entre outros. Estas ferramentas são um grande apoio para os programadores web, uma vez que o JavaScript não tem um compilador associado, estas muitas vezes servem para fazer *debug* da aplicação. O Bataran é um exemplo de uma extensão de apoio web que se associa à *developer tools* do navegador Google Chrome. Esta extensão foi implementada pela empresa Google, para auxiliar a análise de aplicações desenvolvidas em AngularJS. Como foi utilizada para analisar o código do projeto, esta ferramenta é analisada com maior detalhe no capítulo que aborda a fase de desenvolvimento.

Com a banalização do uso da linguagem JavaScript, surgiram muitas bibliotecas de apoio, grande parte gratuitas e com bom nível de maturidade. Estas ferramentas pretendem facilitar a implementação de UI (*User Interface*) poderosas e escaláveis, diminuindo ao máximo o esforço necessário para a interoperabilidade ente os diferentes navegadores web. Cada biblioteca tem as suas próprias funções, com diferente sintaxe e tipo de arquitetura distintos. No entanto, abrangem sempre algumas tarefas comuns, como animações, manipulação DOM, entre outras finalidades.

Nos últimos anos, o JavaScript tem sentindo um desenvolvimento surpreendente. Esta linguagem está mundialmente a superar qualquer outra linguagem de desenvolvimento web. Por este motivo, o número de ferramentas novas para JavaScript também não tem parado de crescer. Existem inúmeras ferramentas, cada uma com lógica de funcionamento diferentes entre si.

Enquanto algumas seguem o conceito de *data-bindings*, como é o caso da ferramenta AngularJS, outras, como Backbonejs, não usam qualquer tipo de *data-bindings*. Cada ferramenta tem particularidades que a distingue e por isso a escolha de qual a melhor só pode ser feita com base na aplicação que se pretende implementar.

Uma forma de analisar as ferramentas de JavaScript é considerar as estatísticas disponibilizadas pelo GitHub, onde se encontram os projetos destas ferramentas de código aberto. Estas estatísticas representam o tamanho, força e impulso que cada ferramenta tem na comunidade web, bem como o estado e atividade dos seus projetos. Considerando as principais ferramentas de código aberto de JavaScript e o número de contribuidores ao longo dos anos, tanto o AngularJS como o Ember.js se destacam, tal como se pode verificar no Gráfico 2.2.

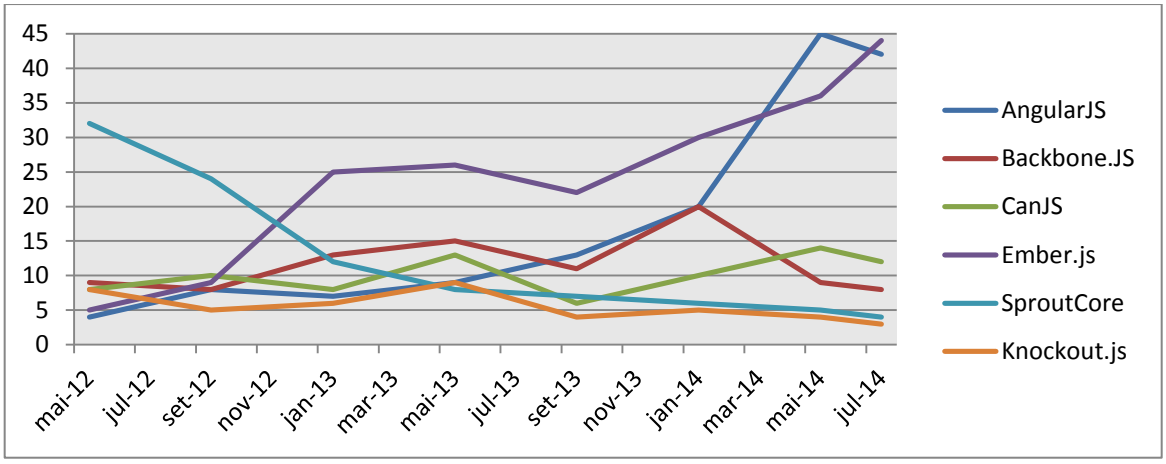

Gráfico 2.2 – Número de contribuidores ao longo do tempo

Analisando o gráfico, percebemos o nível de atividade de cada projeto nos últimos anos e a popularidade que têm na comunidade. Os projetos AngularJS e Ember.js são os que têm mais contribuidores, principalmente nos últimos meses. Ambos sentiram um enorme crescimento e interesse na comunidade web. Em contrapartida, os restantes projetos tendem a não evoluir ao longo dos meses, pois o número de contribuidores também não tem aumentado significativamente. Esta análise serve para determinar o nível de maturidade de uma ferramenta e perceber a sua longevidade. Contudo é importante esclarecer que alguns projetos são geridos por um grupo controlado e pequeno de pessoas, enquanto outros abrem as contribuições para um público amplo e diversificado.

Os elevados números de contribuidores em cada projeto, AngularJS e Ember.js face às demais ferramentas reflete a razão pela qual têm tido tantas atualizações nos seus projetos nos últimos meses. O facto do projeto AngularJS ter tantos contribuidores, fez com que em tão pouco tempo surgissem novas funcionalidades e fossem corrigidos eventuais erros detetados. Ao mesmo tempo que existirem tantos interessados em contribuir para o projeto AngularJS poder significar que é bastante fácil de compreender e por tal de contribuir. A comunidade em geral sente que é

um projeto interessante e que irá crescer no futuro. Neste sentido, pode ser visto como um indicador de que o AngularJS terá provavelmente grande sustentabilidade da comunidade a longo prazo. Contudo é preciso ter em conta que estes números não significam que as ferramentas com mais atividade e maior número de contribuidores sejam melhores que as demais. Para chegar a conclusões mais sustentáveis é necessário analisar com detalhe cada ferramenta verificando quais as funcionalidades suportadas.

Para análise de cada ferramenta foram destacadas 4 características fundamentais: *ui-bindings,* vistas compostas, integração com outras tecnologias e aplicação de filtros. Foram escolhidas estas funcionalidades por serem muito importantes e por não serem suportadas por todas as ferramentas servindo assim de termo de comparação. As 4 características analisadas têm o seguinte significado:

UI *Bindings*: atualização automática da UI quando é detetada a mudança de um objeto observável.

Vistas compostas: Combinação de diferentes componentes de visualização, permitindo também a possibilidade de hierarquia entre todos. Por exemplo: uso de *widgets* de paginação que possam ser reutilizáveis.

Integração com outras tecnologias: permite integração no mesmo projeto do uso de diferentes ferramentas e tecnologias. Por exemplo: uso de jQuery com AngularJS.

Aplicação de filtros: componentes de visualização que exibem objetos que são filtrados com base num determinado critério.

A Tabela 2.1 tem a informação sobre o suporte que é feito por cada ferramenta para cada característica considerada.

	AngularJS	Backbone.js	Ember.js	Knockout.js
UI *Bindings*	✓	X	✓	✓
Vistas Compostas	✓	X	✓	X
Integração com Outras Tecnologias	✓	✓	✓	✓
Aplicação de Filtros	✓	X	✓	✓

Tabela 2.1 – Análise das ferramentas

Com base nesta análise, as ferramentas AngularJS, Ember.js e Knockout.js destacam-se da ferramenta Backbone.js.

Contudo, para além das ferramentas anteriormente consideradas existem outras que já possuem um nível de maturidade muito superior. Como é exemplo o jQuery. Usada por cerca de 61% das 10 mil páginas mais visitadas do mundo, como se pode observar no Gráfico 2.3 [4], o jQuery é das ferramentas JS (JavaScript) mais conhecidas e a mais usada pelos programadores web de todo o mundo. O Gráfico 2.3 agrega informação sobre as percentagens de páginas na internet mais visitadas que foram implementadas utilizando uma ou mais destas ferramentas web, informação obtida a Setembro de 2014. O jQuery é, de longe, a ferramenta mais utilizada e por isso merece ser analisada com maior detalhe.

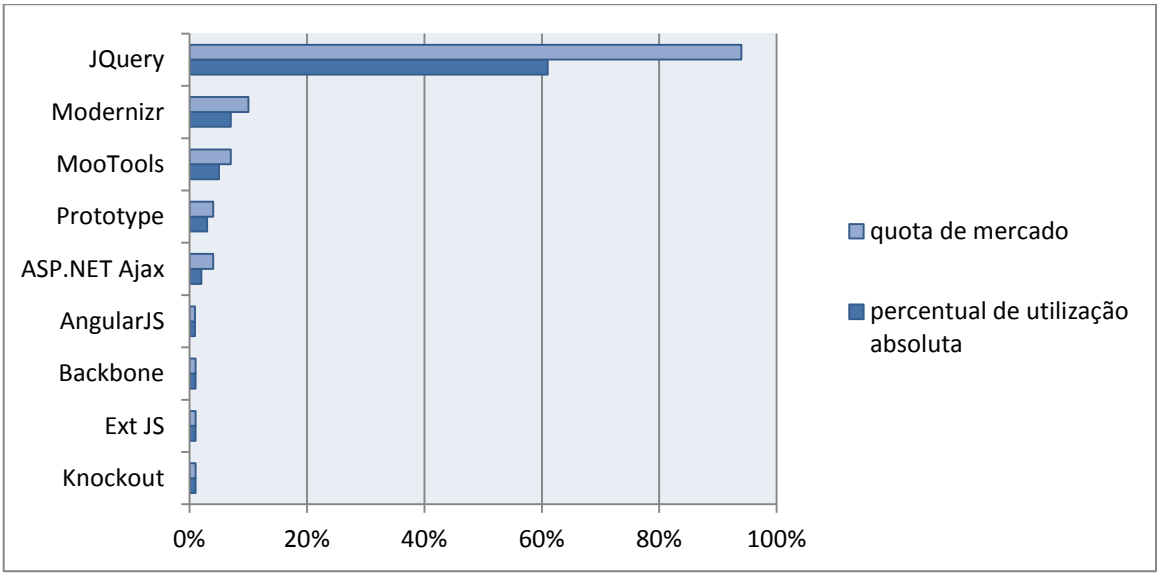

Gráfico 2.3 – Percentagem de páginas web que usam estas ferramentas

Em conclusão, tendo em conta os dados analisados, pode-se considerar que as ferramentas AngularJS e EmberJS, apesar de serem relativamente recentes, estão a ganhar muita força na comunidade de programadores web e provavelmente irão ganhar cada vez mais maturidade no futuro. Mesmo assim o jQuery, por já existir há largos anos, tem um nível muito mais elevado de maturidade. No caso do Backbone é notável o interesse e número de aplicações já implementadas usando esta ferramenta e por isso merece ser considerada para análise na dissertação. Segue de seguida a análise detalhada das ferramentas que se destacam.

2.6.1 JQuery

JQuery[4] é uma ferramenta JavaScript. Surgiu pela primeira vez a Janeiro de 2006, em Nova Iorque, por John Resig e desde então, mudou a maneira de programar web de milhões de pessoas. Por existir há largos anos é considerada uma ferramenta muito estável e com uma comunidade de suporte muito forte.

[4] http://jquery.com/

Esta ferramenta é muito usada por ser fácil de aprender e de utilizar, simplificando a programação JS. Ao longo dos anos, foram criadas novas extensões de apoio para complementar as funcionalidades já existentes do jQuery, o que a torna hoje, muito rica em recursos e a mais extensível. Por exemplo, para automatização de testes de unidade foi criada a extensão QUnit, mas uma das mais conhecidas é a UI jQuery. Este *plugin* é muito usado e útil no desenvolvimento de aplicações web altamente interativas. Esta extensão UI é mais vocacionada para a implementação de funcionalidades relativas a interações com o utilizador, como por exemplo efeitos e animações, *widgets* e temas. Com este *plugin*, muitas funcionalidades são simplificadas, como por exemplo, implementar um formulário com escolha de data de um calendário ou para implementar efeitos em objetos: efeitos de *drag-and-drop*, elementos redimensionáveis, seleção, ordenação, etc.

O jQuery facilita a implementação de aplicações web porque implementa um nível de abstração para o programador. Esta simplifica questões complexas do JS como pedidos Ajax (*Asynchronous JavaScript and XML*) e manipulação DOM. Na prática, aquilo que a ferramenta faz é reunir um conjunto de tarefas comuns que exigem muitas linhas de código JS e envolve-as em métodos jQuery que se podem chamar com uma única linha de código. Esta ferramenta tem as seguintes características:

- Manipulação HTML/DOM
- Manipulação CSS
- Métodos de eventos HTML
- Efeitos e animações
- Pedidos AJAX
- Reutilização de código através de *plugins*

Esta ferramenta é de código aberto, o que também contribuiu para o grande crescimento e popularidade que tem hoje. De notar que existem outras ferramentas JS que incluem alguns métodos do jQuery, como é o caso do AngularJS.

Apesar de ser uma ferramenta em crescimento e com um número elevado de *plugins,* o jQuery, não é aconselhado para implementar projetos de grandes dimensões ou para projetos que se prevê serem escaláveis e extensíveis. Utilizando esta ferramenta existe alguma dificuldade em implementar um projeto modular, com uma clara separação no código entre DOM e JS. A longo prazo, o código fica disperso e difícil de reutilizar. Por norma, esta ferramenta é utilizada para implementações de projetos pequenos, com poucas exigências. O jQuery possibilita a implementação deste tipo de projetos num curto espaço de tempo. Pode também ser usada para implementar *widgets* que serão integrados num projeto de maiores dimensões, ou seja, como ferramenta de complemento.

2.6.2 Ext JS

A ferramenta Ext JS[5] surgiu inicialmente como uma extensão da biblioteca YUI, pela autoria de Jack Slocum. Esta ferramenta é propriedade da Sencha Inc e é um projeto muito ativo. Esta ferramenta JavaScript é utilizada para implementar aplicações web altamente interativas. Para a interação entre cliente e UI, usa técnicas de Ajax, DOM *scripting* e DHTML (*Dynamic* HTML) e permite ao programador desenvolver aplicações UI baseadas numa arquitetura MVC. Para facilitar a implementação de tarefas repetitivas na implementação de aplicações web o Ext JS inclui um conjunto de controlos de formulários baseados em GUI (*Graphical User Interface*) que têm a capacidade de comunicar com o servidor web usando Ajax.

O Ext JS permite a interoperabilidade com outras ferramentas JS tal como as bem conhecidas jQuery e Prototype e tem a vantagem de não estar dependente de nenhuma ferramenta externa. O Ext JS suporta diversas versões de navegadores web, desde as mais antigas até às mais recentes:

- IE6+
- Firefox 3.6+ (PC e MAC)
- Safari 4+
- Opera 11+ (PC e MAC)
- Chrome 11+

A documentação é muito completa, a comunidade é muito ampla e o suporte para os programadores é muito forte. Existem muitos tutoriais de iniciação ao Ext JS e muitos exemplos com código disponibilizado. A maior desvantagem desta ferramenta é não ser de código aberto e os preços serem muito elevados. O preço do pacote da biblioteca mais acessível da versão mais recente Ext JS 4 ronda os 600 dólares.

O Ext JS disponibiliza uma imensidão de funcionalidades muito úteis. Para implementação de tabelas as mais interessantes são a expansão de linhas, alteração da cor da linha com a deteção do rato, ordenação de valores por coluna, aplicação de filtros, omitir colunas da tabela, efetuar o bloqueio de colunas de forma a fixa-las na tabela mesmo que seja feito o *scroll*, existe *data binding* entre a linha da tabela selecionada e os valores apresentados noutro componente. Pode ser feito pesquisa de valores na tabela, implementação de paginação, cabeçalhos agrupados, edição dos valores nas células, entre muitas outras funcionalidades.

[5] http://www.sencha.com/

Na página oficial da ferramenta estão disponíveis imensos exemplos[6], sendo apresentados na Figura 2.3 e Figura 2.4 alguns destes exemplos, nomeadamente de tabelas.

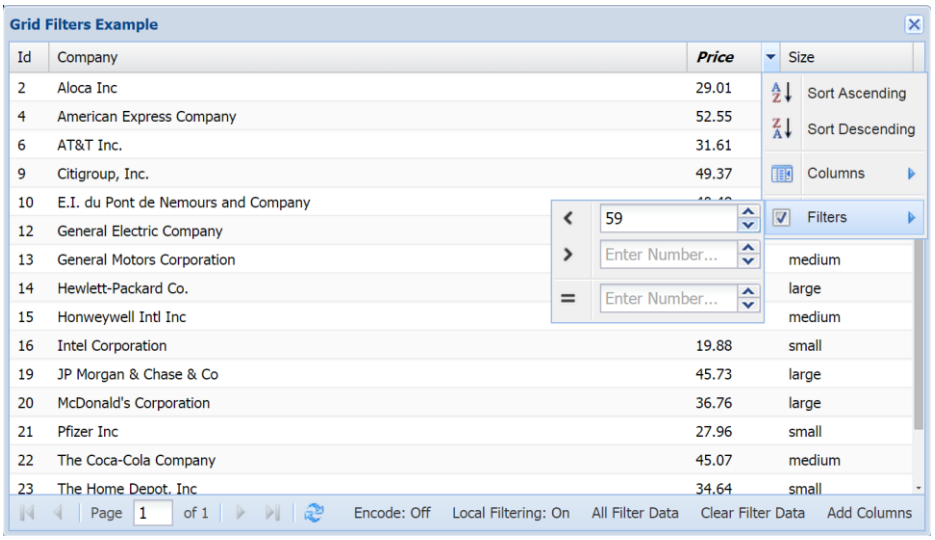

Figura 2.3 – Exemplo de tabela com filtros em Ext JS

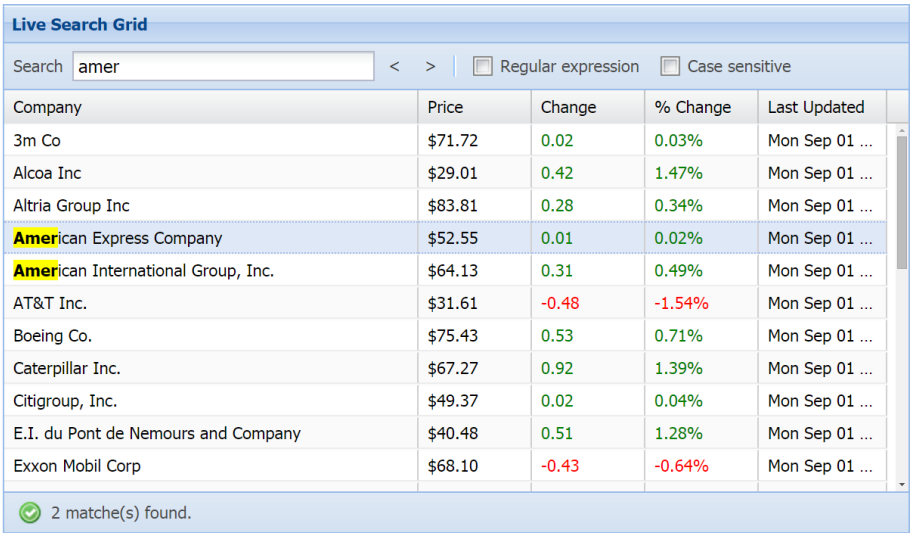

Figura 2.4 – Exemplo de tabela com pesquisa em Ext JS

Esta ferramenta pode ser uma boa opção para projetos e empresas com grande poder económico, principalmente para a implementação de tabelas que tenham de ter muitas funcionalidades. O suporte para esta ferramenta é muito forte e existem muitos exemplos implementados onde é possível obter o código fonte e com um elevado nível de qualidade e flexibilidade para o utilizador.

[6] http://dev.sencha.com/deploy/ext-4.0.1/examples/

2.6.3 AngularJS

AngularJS[7] é uma ferramenta JS que veio revolucionar a implementação de componentes web. Esta ferramenta tem características muito interessantes que permitem implementar código reutilizável e modular. Utilizando esta ferramenta é visível o conceito *loose coupling*, ou seja, uma distanciação entre o componente existente e os outros componentes que o possam vir a utilizar. Esta capacidade permite que a atenção do programador seja vocacionada separadamente: ou para o comportamento da aplicação ou para o seu aspeto.

O AngularJS é umas das ferramentas utilizadas no projeto e por este motivo é necessário uma análise detalhada de algumas das suas principais características [5] no âmbito deste projeto:

- Compatibilidade:

Esta ferramenta funciona muito bem com outras tecnologias e ferramentas, ao contrário de outras que obrigam a um compromisso total do projeto àquela biblioteca. Desta forma é possível implementar um projeto que integre múltiplas aplicações AngularJS e ao mesmo tempo implementar ou integrar widgets e outros componentes implementados com outras ferramentas como por exemplo jQuery ou Ext JS. Desta forma é possível tirar partido do melhor de todos os mundos de cada ferramenta, aumentando a funcionalidade e qualidade final do projeto.

- *Dependency Injection*:

Esta característica presente no AngularJS permite descrever de uma forma declarativa as dependências entre os vários módulos do projeto. Estas dependências podem ser de módulos já existentes e disponibilizados pelo AngularJS como por exemplo o módulo ngRoute ou dependências entre módulos criados e presentes no próprio projeto.

- Testes:

Nota-se que o AngularJS foi desenvolvido a pensar na importância que os testes têm. Os testes devem acompanhar todo o tempo de vida de um projeto. O facto de existir uma separação clara entre comportamento e *templates* em projetos AngularJS, traz grandes vantagens a esse nível porque facilita a implementação dos testes. O AngularJS é composto por vários módulos, um deles para testes é o módulo ngMock, que permite esconder complexidades externas no código dos testes. A implementação de testes também é facilitada pela capacidade da injeção de dependências. Para implementar e executar testes *end-to-end* existe a ferramenta Protractor que foi implementada especificamente para projetos em AngularJS. O Protractor simula as interações do utilizador permitindo assim avaliar qual a qualidade da aplicação AngularJS implementada e deteção de eventuais erros na aplicação.

[7] https://angularjs.org/

Um exemplo prático de um teste *end-to-end* é apresentado de seguida:

```
describe('TODO list', function() {
    it('should filter results', function() {

      // Procura o element com ng-model="user" e associa 'jacksparrown' nele
      element(by.model('user')).sendKeys('jacksparrow');

      // Procura o primeiro botão da página e click nele
      element(by.css(':button')).click();

      // Verifica que existem 10 tarefas
      expect(element.all(by.repeater('task in tasks')).count()).toEqual(10);

      // Introduz 'groceries' no element com ng-model="filterText"
      element(by.model('filterText'))sendKeys('groceries');

      // Verifica que agora só existe 1 item na lista de tarefas
      expect(element.all(by.repeater('task in tasks')).count()).toEqual(1);
    });
});
```

Este tipo de testes pretende verificar se determinado comportamento está a ser feito em conformidade com o esperado. Este exemplo usa métodos da biblioteca Jasmine de JS, também usada neste projeto e que será posteriormente analisada com maior detalhe. Outro tipo de testes igualmente importantes são os testes de unidade. Para estes existe a ferramenta Karma que também será analisada com maior detalhe, por ser uma ferramenta que acompanhou toda a fase de implementação do projeto.

- Diretivas:

As diretivas são um conceito próprio do AngularJS. As diretivas pretendem simplificar a implementação de componentes reutilizáveis e possibilitam esconder a complexa estrutura DOM, CSS e lógica comportamental associada aos componentes. Esta caraterística é uma das principais razões para a escolha desta ferramenta para a implementação do projeto em detrimento de qualquer outra ferramenta. A importante capacidade de criação de diretivas próprias que poderão ser reutilizadas e que conseguem interagir entre si, facilitam a reutilização de código e simplificam a criação e escalabilidade do projeto para implementação de mais componentes web, tornando o código modular e fácil de compreender.

Sem dúvida que as diretivas são a característica de distinção do AngularJS das restantes ferramentas JS. Com a criação de diretivas é possível criar uma nova sintaxe HTML, ou seja, novas *tags* HTML à escolha do programador, sejam elementos ou atributos, ou ambos. A nova sintaxe é especificada pelo programador, com a vantagem desse componente ser reutilizável e esconder toda a lógica comportamental e funcionalidades pretendidas. A possibilidade de criação de novas diretivas é muito poderosa e única pois só está presente na ferramenta AngularJS. A implementação de uma diretiva AngularJS é por exemplo:

```
myModule.directive('myComponent', function(myService) {
      return {
          restrict: 'E',
          controller: function($scope, $attrs, myService) {
              $scope.$on('clickMe', function() {
                  $scope.message = 'Directive: ' + myService.message;
              });
          },
          replace: true,
          template: '<input>'
      };
  });
```

Para utilizar esta diretiva basta no entanto escrever uma única linha de código:

```
<my-component ng-model="message"></my-component>
```

Criando uma aplicação como um conjunto de componentes web torna-se incrivelmente fácil adicionar, atualizar ou eliminar funcionalidades conforme necessário. No entanto é primeiro necessário alguém implementar os componentes com as funcionalidades necessárias.

- MVC:

O AngularJS incorpora os princípios básicos por detrás do padrão original de design de software MVC, este conceito por ser tão importante na arquitetura e estrutura do projeto é abordado no capítulo 3, que analisa isso mesmo, a arquitetura e padrões de design implementados no projeto.

- *Two-data-binding*:

Aquando da comparação das diferentes ferramentas web, foi abordada a funcionalidade *UI Bindings*. No caso do AngularJS, este implementa o *two-data-binding*, ou seja, as alterações feitas pelo utilizador na aplicação são automaticamente detetadas do lado da aplicação e vice-versa. Esta característica evita escrever uma quantidade considerável de código clichê. Internamente, a ferramenta faz uma sincronização entre DOM e modelo.

Aqui está um exemplo simples que demonstra como vincular um valor de entrada para um elemento DOM:

```
<!doctype html>
<html ng-app>
  <head>
    <script src="http://code.angularjs.org/1.3.2/angular.min.js"></script>
  </head>
  <body>
    <div>
      <label>Name:</label>
      <input type="text" ng-model="myName" placeholder="Enter your name">
      <hr>
```

```
      <h1>Hello, {{myName}}!</h1>
    </div>
  </body>
</html>
```

Usando os caracteres {{ }} ou usando a diretiva do AngularJS “ng-bind” torna-se muito simples de implementar o *two-data-binding*.

Esta ferramenta está a ganhar cada vez mais interesse na comunidade e o número de projetos no GitHub implementados usando AngularJS também está a aumentar [6]. O AngularJS tem grandes potencialidades de crescimento e tem fortes características que a podem tornar na próxima escolha de eleição pela maioria dos programadores web. Esta característica torna o AngularJS uma ferramenta muito atrativa para a criação de componentes web, como é o caso deste projeto. Por todos estes motivos esta ferramenta é a adotada para o projeto.

2.6.4 BackboneJS

O BackboneJS[8] foi desenvolvido por Jeremy Ashkenas também autor da conhecida ferramenta CoffeeScript. BackboneJS é uma ferramenta de JavaScript baseada no padrão de software MVP (*Model-View-Presenter),* com uma interface JSON RESTful (RESTful é o nome utilizado para denominar os sistemas que seguem os princípios REST - *Representational State Transfer*). Esta ferramenta é relativamente recente, lançada a Outubro de 2010 e é conhecida por ser leve e projetada para implementar aplicações web SPA (*Single-Page Application*). Esta ferramenta também pode ser usada para manter diferentes partes da aplicação sincronizadas, como por exemplo vários clientes com o servidor. O BackboneJS tem um grande problema associado, a sua dependência com uma outra ferramenta de JavaScript, nomeadamente o Underscore.js

Tendo em conta a dimensão do projeto BackboneJS no GitHub [7], este não tem muitos erros por corrigir. É uma ferramenta que está a ganhar maturidade e é claramente muito popular no GitHub. Existem muitas empresas, tanto grandes como pequenas que decidiram escolher esta biblioteca para implementação de projetos, sejam eles de grandes ou pequenas dimensões. Falamos por exemplo da loja *online* da Sony Entertainment Network ou das ferramentas DropTask e Kanban Tool por exemplo, também desenvolvidas usando BackboneJS [8].

Contudo, esta ferramenta não permite implementar componentes web e em comparação com as ferramentas anteriormente analisadas, esta não contém funcionalidades que se destaquem das demais.

[8] http://backbonejs.org/

2.7 Bibliotecas Gráficas

Agora que já foram analisadas e escolhida a melhor ferramenta JS é necessário analisar e escolher a melhor biblioteca para a implementação dos gráficos a implementar no projeto.

Existem imensas bibliotecas gráficas na linguagem JavaScript para representação e manipulação de informação e por isso serão analisadas somente as mais populares. Em tempos também existiram muitas bibliotecas em Flash com o mesmo intuito mas tem-se vindo a verificar que foram descontinuadas pelas suas empresas proprietárias, como é o caso da biblioteca FusionCharts que está a preparar uma nova versão para a linguagem JS e a descontinuar a sua anterior versão baseada em Flash. Outro exemplo é a biblioteca amcharts que também está a migrar para a linguagem JS por esta ser a linguagem tão procurada pelos programadores web da atualidade. O Flash está decididamente condenado a pertencer ao passado e não ao futuro e por isso somente serão analisadas bibliotecas gráficas baseadas em JS usadas para a criação de gráficos ricos em interações, manipulação de dados e outras funcionalidades importantes neste tipo de componentes.

2.7.1 HighchartsJS

HighchartsJS[9] é uma API desenvolvida em JavaScript para implementação de gráficos web. Esta API foi desenvolvida em 2006 pela companhia Highsoft AS e desde então tem evoluído bastante e ganho cada vez mais entusiastas. Existe uma grande preocupação no apoio e suporte aos programadores, uma vez que existe na página oficial da ferramenta muita documentação e exemplos práticos implementados. Além disso, existe um fórum muito ativo para resposta a questões técnicas de implementação e suporte relativamente a erros detetados na API.

Com o tempo, o HighchartsJS, tem vindo a crescer, sendo disponibilizados novos módulos como por exemplo o *export* e *print* que tal como o nome em inglês indica, permite exportar os gráficos e/ou imprimi-los. Recentemente foi implementado o módulo *data*, que permite que os dados representados num gráfico sejam os dados extraídos de uma determinada tabela. Esta tabela tem de ser definida e implementada anteriormente pelo programador, sendo posteriormente feita a extração dos seus dados, automaticamente, por este módulo.

Esta ferramenta permite a criação de múltiplos tipos de gráficos, desde a implementação de gráficos de linhas, de área, bem como gráficos de colunas extensíveis, mistos, entre outros tipos. Alguns exemplos existentes na página oficial da API são visíveis na Figura 2.5.

[9] http://www.highcharts.com/

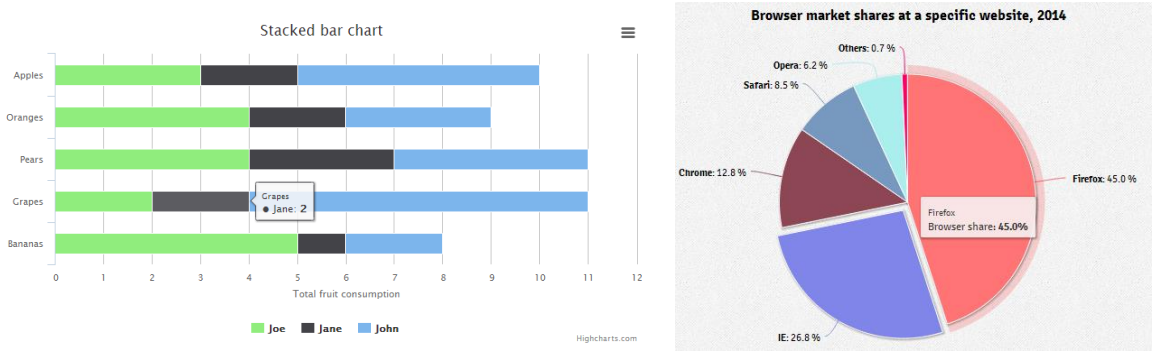

Figura 2.5 – Exemplos de gráficos Highcharts[10]

Esta biblioteca é de código aberto desde que não seja para uso comercial. Felizmente a empresa Nokia Networks detém uma licença para o uso desta ferramenta e neste caso não existe qualquer tipo de restrição para a usar. Como num futuro próximo o objetivo deste projeto é estar integrado em produtos da empresa Nokia Networks é extremamente importante ter em conta que todas as ferramentas usadas têm de ser de código aberto ou ter como neste caso uma licença paga.

Highcharts é a escolhida para usar no projeto da dissertação para apoio no desenvolvimento de gráficos web usando a linguagem JS.

2.7.2 D3

O D3[11] está para *Data-Driven Documents*. Esta biblioteca permite mais do que a criação de gráficos aos quais estamos habituados, esta permite um nível muito elevado de manipulação e representação dos dados, possibilitando implementar visualizações de dados menos usuais. Esta biblioteca está constantemente a evoluir, com lançamentos de novas versões.

Esta biblioteca JavaScript permite a manipulação de documentos com base em dados. É uma ferramenta que tira o máximo partido das potencialidades do HTML, SVG e CSS permitindo a criação de diferentes tipos menos normais de gráficos e sempre com um nível muito elevado de animações. Alguns desses exemplos são apresentados na Figura 2.6.

[10] http://www.highcharts.com/demo
[11] http://d3js.org/

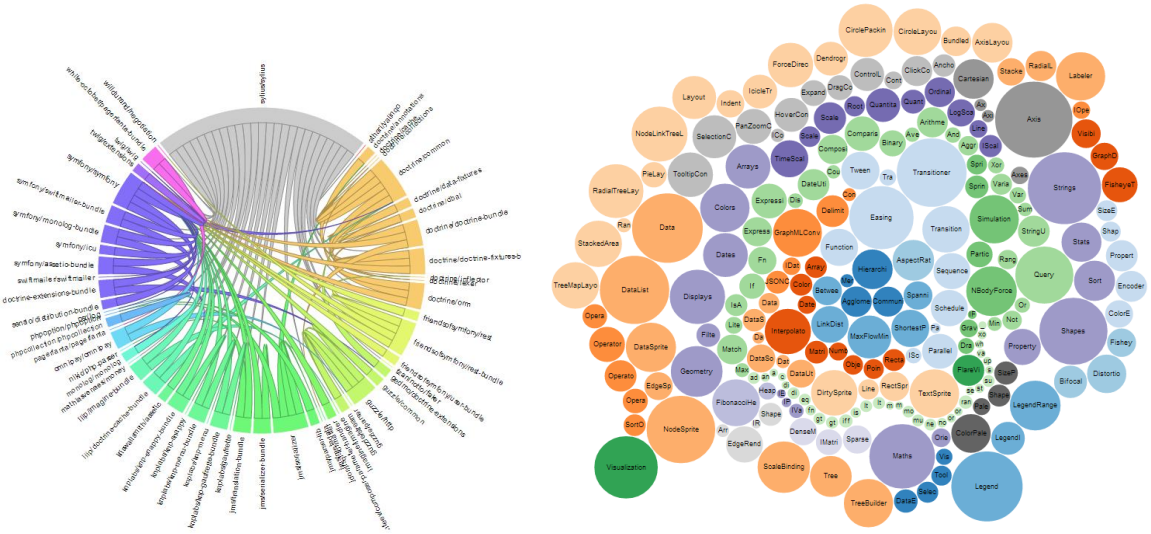

Figura 2.6 – Exemplos de gráficos D3[12]

Apesar das grandes capacidades desta biblioteca, o foco da dissertação passa por aumentar, ao máximo, o nível de interatividade de gráficos web mais comuns e não a implementação de outros tipos de representação de dados mais estranhos para o qual esta API também está definida.

2.7.3 Google Chart Tools

O Google Chart Tools[13] é uma ferramenta gráfica de JavaScript de código aberto, lançada em 2007, fácil de usar e compreender, desenvolvida pelos programadores da empresa Google. Os gráficos implementados são muito simples o que torna esta ferramenta muito leve e com bom desempenho. Usando o Google Charts Tools é possível, em pouco tempo, criar gráficos para representação de dados, mas o resultado visual dos gráficos não é muito atrativo nem dinâmico como outras ferramentas anteriormente analisadas.

Esta ferramenta só é uma boa opção de escolha quando o *design* final dos gráficos não for uma preocupação e os gráficos criados sejam só para apresentação de dados em alturas de maior urgência, como por exemplo para colocar em apresentações ou projetos pequenos internos para os utilizadores da empresa e não para integrar projetos que se pretenda vender. Alguns exemplos implementados usando esta ferramenta são apresentados na Figura 2.7.

[12] https://github.com/mbostock/d3/wiki/Gallery
[13] https://developers.google.com/chart/

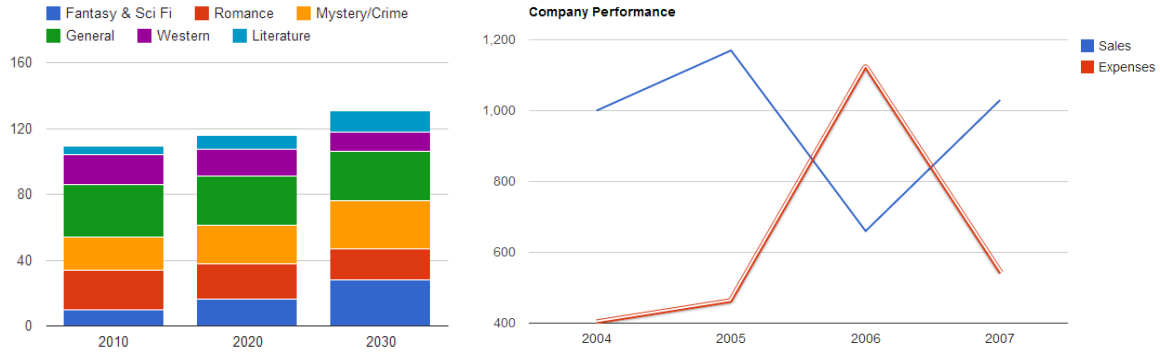

Figura 2.7 – Exemplos de gráficos Google Chart Tools[14]

Como se pode observar pelas figuras anteriores, os gráficos tem uma apresentação muito pobre, com aparência demasiado básica. Esta biblioteca nos últimos anos não tem vindo e evoluir pois também não teve muito sucesso como ferramenta gráfica web. Mesmo assim, precisa de evoluir muito ainda para conseguir atrair a comunidade de programadores web para implementação de gráficos de elevada qualidade.

2.8 Síntese

Neste capítulo foram abordadas a várias linguagens alternativas para a implementação de componentes web. Chegando-se facilmente à conclusão que as linguagens HTML5, CSS e JS são as melhores nos dias de hoje para projetos web e por isso as escolhidas. Foram também analisadas neste capítulo tanto as ferramentas JS para implementação dos componentes, bem como de ferramentas gráficas específicas para a implementação de gráficos em JS. Conclui-se que existem muitas ferramentas e bibliotecas JS à escolha, mas tanto a ferramenta AngularJS como a biblioteca Highcharts são as que melhor correspondem às necessidades do projeto a desenvolver e por este motivo, ambas escolhidas.

[14] https://developers.google.com/chart/interactive/docs/gallery

3 Arquitetura do Projeto

Neste capítulo será analisada a arquitetura do projeto bem como os requisitos a ter em conta. Serão também analisadas as ferramentas que fizeram parte de toda a fase de implementação do projeto, tanto para implementação da lógica dos componentes bem como das ferramentas próprias para implementação dos testes. Neste mesmo capítulo e por pertenceram à arquitetura do projeto, serão também descritos os padrões e conceitos web inerentes à implementação de projetos web deste tipo.

3.1 Requisitos do Projeto

Relembrando que o objetivo do projeto é a implementação de componentes web para visualização e manipulação de informação é importante definir como serão recebidos esses mesmos dados e qual o seu formato. Esta questão representa um dos requisitos do projeto. Todos os requisitos do projeto são apresentados de seguida:

- Formato de dados JSON
- Linguagens e Tecnologias inovadoras
- Componentes web
- *Cross-Browser*
- Metodologias ágeis: TDD, Agile e Scrum
- Look&Feel
- Possibilidade de alcançar os dispositivos móveis num futuro próximo

Formato de dados JSON:

Os dados são recebidos em ficheiros JSON, que têm uma estrutura JSON previamente acordada entre quem implementa os serviços web e o projeto, o qual vai consumir este serviço. O servidor de *Web Services* com todos os serviços necessários para este e outros projetos é neste momento o projeto de dissertação de outro elemento na empresa. O projeto, por existir há muito pouco tempo, está num estado muito embrionário, neste sentido foi necessário implementar uma solução alternativa que passa por simular um servidor que envie estes mesmos ficheiros, com a

mesma estrutura e dados simulados. Os ficheiros são obtidos depois de efetuado um pedido HTTP, pela aplicação cliente, a um servidor.

Contudo, convém perceber o que é o formato JSON para que se consiga manipular a informação contida nos ficheiros recebidos.

O formato JSON é um formato de dados, muito utilizado nos dias de hoje. Este formato por ser tão leve e fácil de interpretar tanto pelo ser humano como pela linguagem JavaScript é a melhor escolha para troca de informação web.

Um objeto JSON é basicamente um par nome e valor, separado por dois pontos e contido entre parênteses.

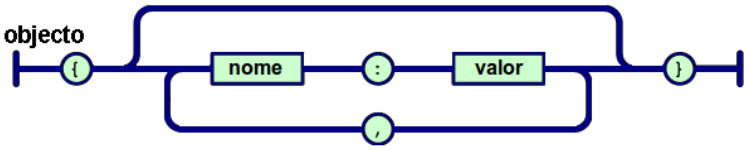

Figura 3.1 – Objeto JSON

No campo valor do objeto JSON podemos ter diversos tipos valores:

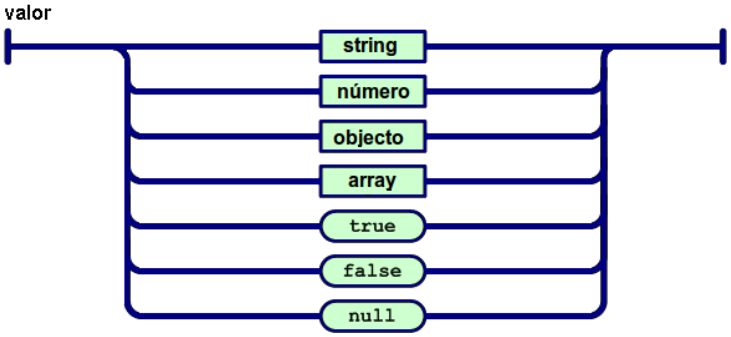

Fig. 3.1: Valores possíveis do objeto JSON

É possível então formar *arrays* de valores ou então *arrays* de objetos JSON.

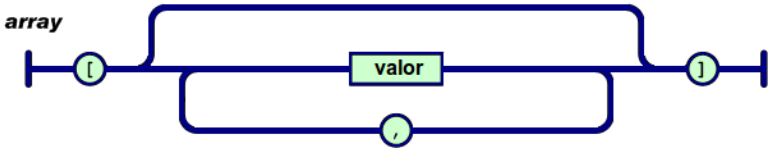

Figura 3.2 – Array JSON

O JSON traz mais desempenho a uma aplicação web do que o XML e por isso o JSON é o formato de dados usado no projeto.

Linguagens e Tecnologias inovadoras

Para o desenvolvimento do projeto, recorreu-se a várias ferramentas web, tanto ferramentas para implementação da lógica do projeto, como ferramentas para implementação de testes, manipulação de pacotes web, entre outras. Foram ainda utilizadas vários tipos de linguagens de programação web, nomeadamente, HTML5, CSS e JavaScript. Um dos muitos objetivos que nos propusemos atingir com este projeto passou pelo desenvolvimento do mesmo com base em tecnologias inovadoras mas com capacidade de permanência e suporte no futuro.

O projeto tem de ser baseado em padrões e arquiteturas web modernos. Conceitos como *single-page-aplication, model-view-controller*, entre outros, são importantes para o futuro de uma aplicação web que pretende corresponder às expectativas atuais dos seus utilizadores. Do ponto de vista do programador, usar ferramentas capazes de implementar estes conceitos tornam a aplicação escalável, reutilizável e de código modular. Do ponto de vista do utilizador, linguagens como HTML5, CSS3, JavaScript e outras tecnologias possibilitam uma experiência e interatividade muito positiva para o utilizador, permitindo que a aplicação tenha um grande nível de usabilidade.

- Componentes web:

O principal requisito passa pela implementação de componentes web para apresentação e manipulação de dados em gráficos e tabelas. Os componentes web permitem ser reutilizáveis, daí a sua grande importância na comunidade web nos dias de hoje. Os componentes web implicam que depois de implementados, estes possam ser utilizados e integrados em qualquer produto na empresa e mesmo por pessoas com poucos conhecimentos de HTML5, CSS ou JS.

- *Cross-Browser*

O projeto tem de conseguir executar nos diferentes navegadores web da atualidade. Por causa da linguagem HTML5 não ter ainda uma recomendação W3C conduz a dilemas de suporte de funcionalidades. Este dilema traz grandes problemas de implementação na aplicação web porque é necessário esta ser compatível com diferentes navegadores web. Surge assim o conceito cross-browser: capacidade de determinada aplicação ou página web, *scripts client-side* ou código HTML suportar múltiplos navegadores web. Para ser possível o suporte de vários navegadores torna-se imprescindível ter forte conhecimento sobre as novas funcionalidades HTML5 e quais são suportadas pelos navegadores web nas versões que se tem de suportar.

Um dos requisitos da empresa referente ao *cross-browser* é, felizmente, só ser necessário dar suporte às versões IE9+. As versões mais recentes do IE (*Internet Explorer*) têm maior compatibilidade com as recentes tecnologias do HTML5 e do CSS3 ou passo que as versões mais antigas simplesmente não as suportam. Por exemplo, as versões IE8- não suportam por completo o CSS3.

Por ser necessário dar suporte aos vários navegadores web da atualidade é importante compreender o seu nível de suporte com as novas funcionalidades do HTML5 e CSS3. De seguida é apresentada uma tabela, Tabela 3.1 que ajuda a compreender o suporte dado por cada navegador.

	Safari	Chrome	Opera	Firefox	IE9
Local Storage	✓	✓	✓	✓	✓
Histórico de Sessão	✓	✓	✓	✓	✓
Aplicações *Offline*	✓	✓	X	✓	X
Novos tipos de campos	✓	✓	✓	X	X
Form: Autofocus	✓	✓	✓	X	X
Form: Autocomplete	X	X	✓	X	X
Form: Required	✓	✓	✓	X	X
Vídeo, Áudio e Canvas	✓	✓	✓	✓	✓
Media Queries	✓	✓	✓	✓	✓
WebSockets	✓	✓	✓	✓	X

Tabela 3.1 – Compatibilidade e suporte dos navegadores

Para tentar detetar eventuais erros de suporte são executados testes *end-to-end* para todas as versões que se tem de suportar, pois só assim é realmente garantido que existe suporte. Contudo, o esforço e tempo necessário para executar todos os testes, para todas as versões que se tem de suportar são enormes. Na prática seria, considerar o navegador IE9 de 64bits para Linux, IE9 32bits para Linux, IE9 64bits para Windows e afins e por isso é uma tarefa impraticável, mesmo em ambiente empresarial. No projeto optou-se por testar a aplicação nos 5 navegadores mais populares da atualidade: IE, Opera, Mozilla Firefox, Opera e Safari. Para isso foram instalados drivers para cada um destes navegadores web, de forma a executar os testes sobre eles e assim testar as funcionalidades da aplicação. Com isto, falamos agora de outro requisito do projeto, a necessidade de implementar testes de unidade e testes *end-to-end*, seguindo a metodologia TDD.

Metodologias ágeis: TDD, Agile e Scrum

A metodologia TDD permite aumentar a concentração do programador para que somente sejam implementadas as funcionalidades necessárias para que o código passe nos testes. Esta metodologia por ser tão importante em toda a vida do projeto é analisada com maior ênfase no

capítulo de desenvolvimento. TDD implica a implementação de testes de unidade e testes *end-to-end* que devem acompanhar todo o desenvolvimento do projeto.

As metodologias Agile e Scrum permitem um acompanhamento periódico da evolução e funcionalidades do projeto. Estas metodologias permitem detetar, numa fase inicial, eventuais erros ou alterações necessárias, evitando assim maior perda de tempo e esforço do que usando as convencionais metodologias de desenvolvimento de projetos.

Look&Feel

Outro requisito muito importante deste projeto passa por manter o mesmo *look and feel* de todos os outros produtos desenvolvidos na Nokia Solutions and Networks. O *look and feel* da empresa NSN é o Orange Touch 2 mas como já referido e como seria de esperar num projeto empresarial, o *look and feel* passou a ser o Ocean Touch, por agora ser um produto propriedade da Nokia Networks. Como existe um grande nível de independência entre os componentes web implementados e *look and feel* utilizado, a alteração foi possível efetuar ainda em tempo útil desta dissertação e que será analisada no próximo capítulo, referente à implementação do projeto.

Possibilidade de alcançar os dispositivos móveis num futuro próximo

Sabendo que os dispositivos móveis estão a ter enorme aderência em todo o mundo é normal espectar que os componentes web deverão num futuro próximo poder ser integrados neste tipo de dispositivos. Para a implementação do projeto foi necessário ter este pensamento em conta de forma a facilitar no futuro esta transição. Para isso, as ferramentas e linguagens escolhidas tiveram de passar neste requisito. Usando aplicações específicas foi possível verificar que os componentes web implementados conseguem realmente executar num dispositivo móvel. Esta implementação será exposta no capítulo de desenvolvimento do projeto.

Com todos os requisitos definidos, ferramentas JS e linguagens web escolhidas, falta agora descrever qual o IDE e outras ferramentas utilizadas para implementação e gestão de toda a complexidade de um projeto de componentes web.

3.2 Ferramentas de Desenvolvimento

Para a implementação de componentes web em AngularJS é necessário conseguir gerir todas as dependências necessárias no projeto, bem como seguir uma estrutura e lógica comum em projetos AngularJS. Igualmente importante são as ferramentas de testes que têm de executar todos os testes implementados. Para tal é necessário implementar configurações e instalar ferramentas típicas para ambientes de desenvolvimento web. Também importantes são as

ferramentas de análise de código e relatórios com a percentagem de código com cobertura de testes. Passamos então à análise deste tipo de ferramentas de desenvolvimento.

3.2.1 Yeoman, Grunt, Bower e Node.js

Neste tipo de projetos web com alguma dimensão é importante usar ferramentas de apoio ao desenvolvimento da aplicação para aumentar o desempenho e eficiência no desenvolvimento web.

A ferramenta Yeoman[15] serve para ajudar a inicializar e a construir a estrutura inicial do projeto. Com esta ferramenta são automaticamente definidas algumas configurações e as tarefas mais importantes para o lançamento do servidor de desenvolvimento usando o Grunt. O Yeoman permite também gerir eventuais dependências da ferramenta Bower que venham a ser necessárias no projeto.

A ferramenta Grunt[16] permite lançar servidores e definir comandos para executar determinadas tarefas das ferramentas incluídas no projeto. Esta ferramenta implica a configuração do ficheiro: Gruntfile.js, que contém todas as configurações e tarefas. No ficheiro Gruntfile.js foram feitas configurações de modo a, por exemplo, ser possível lançar, através de um simples comando na linha de comandos, a aplicação web num servidor local. Este servidor tem a capacidade de detetar alterações feitas no código e em tempo real fazer o *refresh* automático da página web ficando visíveis para o programador as modificações por ele implementadas. Foram também efetuadas configurações no ficheiro Gruntfile.js do Grunt que permitem o lançamento, através da linha de comandos, de um servidor para os testes de unidade usando o Karma e um servidor para os testes *end-to-end* usando o Protractor. O servidor de testes de unidade também foi configurado de modo a serem executados todos os testes de unidade cada vez que é detetada uma alteração no código. Ambas as ferramentas Karma e Protractor serão analisadas com mais detalhe posteriormente.

A ferramenta Bower[17] é usada para efetuar a gestão das dependências dos pacotes. Deste modo, não é necessário o programador instalar os pacotes e gerir manualmente os scripts. O Bower permite adicionar, remover e atualizar pacotes, bem como criar os nossos próprios pacotes Bower. Esta ferramenta tem como ficheiro obrigatório de configuração o bower.js que contém a versão do projeto, o nome, as dependências com outros pacotes, etc. No entanto, existe também o ficheiro .bowerrc que por exemplo pode conter a localização onde se pretendem instalar as dependências necessárias ao projeto.

[15] http://yeoman.io/
[16] http://gruntjs.com/
[17] http://bower.io/

Por causa de muitas destas ferramentas serem *plugins* do Node.js[18] é necessário, em primeiro lugar, instalar o Node.js. O npm[19] (*Node Packaged Modules*) é como o próprio nome indica, o gestor de pacotes do Node.js. O npm implica a configuração do ficheiro package.json, que contém todas as dependências e *plugins* pretendidos na aplicação web. Estas dependências são diferentes das dependências do Bower pois são *plugins* específicos do Node.js, não existindo nas dependências do Bower.

Na implementação inicial do projeto algumas destas ferramentas só tinham suporte para plataformas Linux mas devido à enorme popularidade que ganharam nos últimos meses, todas estas ferramentas passaram a ter suporte para qualquer sistema operativo da atualidade.

Recapitulando, Yeoman serve para a criação inicial da estrutura do projeto, Grunt para lançamento de servidores e execução de tarefas, Bower como gestor de dependências de módulos bower e npm como gestor de dependências de módulos Node.js.

3.2.2 WebStorm by JetBrains

O WebStorm[20] é o IDE mais recomendado para o desenvolvimento de aplicações web. Este IDE reconhece as linguagens JavaScript, CSS e HTML, permitindo a edição e *refactoring*, com *auto-complete* e análise de código em tempo real, deteta erros de sintaxe e alguns erros lógicos, dando imediatamente dicas para os erros detetados, entre outras funcionalidades comuns existentes nos IDEs a que estamos habituados a trabalhar noutras linguagens. Inclusive, no livro "AngularJS" da autoria de Brad Green e Shyam Seshadri, ambos muito ligados ao mundo do AngularJS, recomendam este IDE dizendo: "WebStorm by JetBrains offers one of the most comprehensive web development platforms in recent times.", ou seja, o WebStorm é a plataforma web mais abrangente dos últimos tempos para desenvolvimento web [9].

Este IDE permite a integração com uma infinidade de bibliotecas web e *plugins* que servem para apoio no desenvolvimento web mesmo de tecnologias mais recentes, tal como Dart, Karma, Jasmine, AngularJS, Node.js, CSS3, HTML5, RequireJS, entre outros e é por isso considerado o IDE mais inteligente da atualidade de JavaScript.

A nível de integração de CVS este IDE suporta vários tipos incluindo os 3 mais conhecidos: Git, SVN e Mercurial. É *cross-platform*, ou seja, tem suporte para Linux, Windows ou Mac OS e permite a integração desde a raiz do projeto com ferramentas web muito úteis, tal como Grunt, Bower e Node.JS que são usadas neste projeto.

[18] http://nodejs.org/
[19] https://www.npmjs.org
[20] https://www.jetbrains.com/webstorm/

Por todos estes motivos o IDE WebStorm foi o escolhido para a implementação dos componentes web.

3.2.3 AngularJS Batarang

O AngularJS Batarang[21] é um *plugin* que serve como ferramenta de apoio no desenvolvimento de projetos implementados com o AngularJS. Esta extensão disponível de forma grátis é associado ao navegador web Google Chrome e serve para *debug*, medição de desempenho e analisar aspetos específicos da aplicação, como questões de dependências entre os módulos, hierarquia e alcance/*scope* dos objetos.

Na Figura 3.3 é possível observar quais os módulos criados e as dependências entre si, ou seja, os módulos que são injetados para cada módulo. Alguns módulos são específicos do AngularJS, nomeadamente os que têm o nome iniciado com o carater "$", os restantes módulos foram implementados para o projeto.

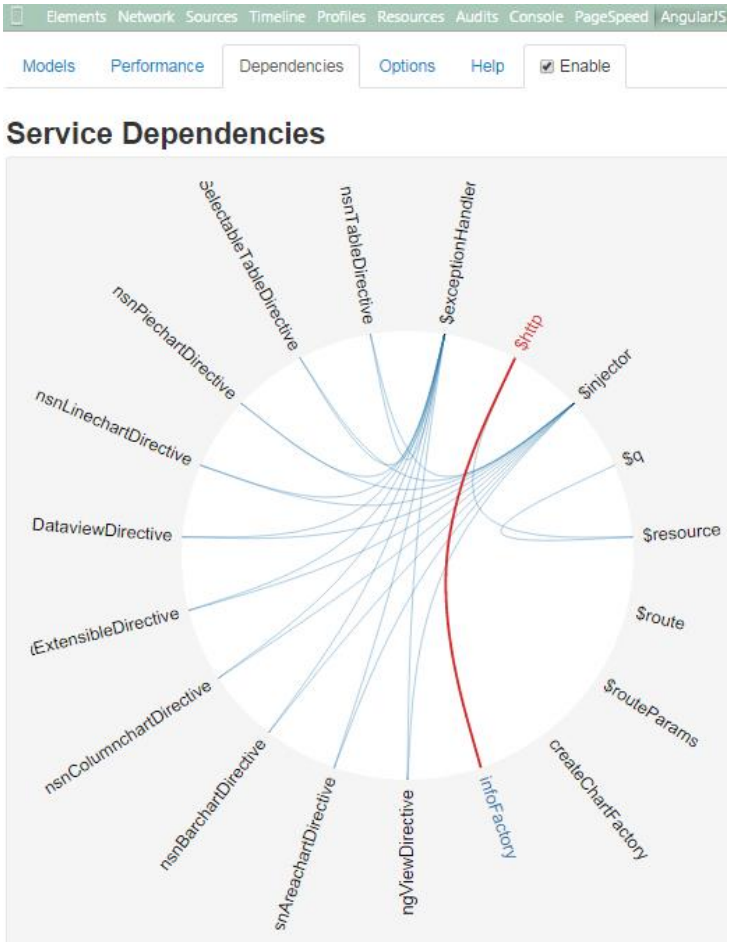

Figura 3.3 – Batarang na vista das dependências

[21] https://chrome.google.com/webstore/detail/angularjs-batarang/ighdmehidhipcmcojjgiloacoafjmpfk

Analisando a Figura 3.3, o módulo “infoFactory” tem como dependência um módulo do AngularJS. O $http que serve para efetuar pedidos HTTP a um servidor para se conseguir obter os dados JSON que são externos ao projeto. Note que neste caso está a vermelho porque com o uso do rato é possível destacar melhor as dependências ficando as ligações entre os módulos a vermelho.

Como a ferramenta AngularJS é do tipo “*two-way data binding*” torna-se muito importante saber os valores das variáveis de cada *scope* em tempo real e qual a hierarquia entre eles. Na vista “Models” da ferramenta Batarang é possível analisar a estrutura hierárquica das variáveis e os seus valores.

Tendo o projeto alguma complexidade associada torna-se fundamental compreender o seu nível de desempenho e onde é que este despende maior parte do tempo de execução. Com o Batarang consegue-se de forma fácil e rápida obter *feedback* das zonas de código da aplicação que precisam ser melhoradas de forma a melhorar o desempenho geral. Para tal existe a vista “Performance”, como é possível ver na Figura 3.3 anteriormente apresentada.

Como o Batarang é somente uma ferramenta de análise em tempo real do código implementado e serve somente como pequeno apoio de *debug* não existe necessidade de detalhar mais a análise a esta ferramenta.

3.2.4 JSHint

O JSHint[22] é uma ferramenta específica para código JavaScript que analisa o código e ajuda a detetar erros ou potenciais problemas no código. O JSHint depois de analisar o código cria um relatório com métricas importantes para o programador.

No projeto, esta ferramenta foi instalada como *plugin* do Grunt e tem como ficheiro de configuração o .jshintrc. Neste ficheiro o programador configura quais as regras que quer que sejam analisadas e que valores máximos permitidos. Por exemplo, definir o valor máximo da complexidade que o código pode ter, a profundidade máxima das funções, o número máximo de argumentos que as funções podem receber, etc.

Com o uso da ferramenta JSHint foi possível corrigir erros básicos como faltas de ; ou de {} mas o mais importante são as métricas. Como por exemplo obter a complexidade ciclomática de trechos de código.

Vamos por exemplo analisar um dos módulos existentes no projeto. O módulo “`nsnChartColumn`” tem implementado várias diretivas para os diferentes tipos de gráficos de

[22] http://www.jshint.com/

colunas. Somente é apresentado um pequeníssimo trecho deste módulo porque o importante analisar neste caso é o relatório com as métricas obtidas relativamente a este módulo.

```
(funcion () {
      “use strict”;
angular.module(“nsnChartColumn”, [‘chartServices’])
      .directive(‘nsnColumnchart’, function () {
            return {
                  restrict: ‘AE’,
                  transclude: true,
                  scope: {
                        filejson: ‘@’,
                        combined: ‘@’,
                        data: ‘@’,
                        theme: ‘@’,
                  },
                  templateUrl: ‘templates/columnChart.html’,
                  (…)
```

Usando a ferramenta JSHint, para este módulo foram obtidas as seguintes métricas:

- Existem 22 funções;
- A função com maior assinatura recebe 3 argumentos, sendo a média 0.5;
- A função maior tem 10 declarações dentro dela, sendo a média de 2;
- A função mais complexa tem complexidade ciclomática 4, sendo a média 1.

Neste módulo estão implementadas as diretivas para os gráficos de colunas simples e extensíveis.

Como o código do projeto está bem dividido e as funções implementadas estão bem estruturadas, a complexidade deste módulo é relativamente baixa. O módulo que tem maior nível de complexidade é o que tem o processamento dos dados recebidos, ou seja, o módulo com a implementação dos serviços internos ao projeto. Os dados recebidos têm de ser manipulados de forma a seguirem a estrutura interna esperada pela ferramenta Highcharts. Nestes casos e na introdução dos dados nas tabelas existe a necessidade de aceder aos objetos dos vários índices de *arrays* e a percorrer um *array* usando cliclos *for* dentro de outro ciclo *for*. Este tipo de complexidade deve ser evitado sempre que possível.

Em conclusão, esta ferramenta é então muito útil para detetar pequenas violações de código bem como detetar eventuais módulos que precisem ser reimplementados para minorar o seu nível de complexidade ciclomática.

3.3 Ferramentas de Testes: Protractor, Karma e Jasmine

3.3.1 Jasmine

O Jasmine[23] é uma biblioteca para implementação de testes em JavaScript. Pode ser utilizada para implementação tanto de testes de unidade, como para testes *end-to-end*, ou outros. É uma ferramenta que não se baseia no DOM, nem nos navegadores, ou qualquer outra estrutura de JavaScript. Por este motivo, pode ser utilizada para projetos Node.js, ou aplicações web, ou qualquer outra tecnologia onde possa ser executado JavaScript.

Esta ferramenta surgiu em Setembro de 2010, é *cross-platform* e tem uma comunidade de programadores ativos. Existe documentação *online*, com exemplos de como implementar diferentes casos de testes de unidade. Um exemplo de um teste unitário é apresentado de seguida:

```
describe("Unit Tests: Directives", function() {
  it("should have at least one html tag div", function() {
    var div = element.find('div');
    expect(div.length).toBeGreaterThan(0);
  });
});
```

Como se pode analisar com base no código anterior, os métodos do Jasmine são por si muito descritivos e explícitos, facilitando a compreensão do código dos testes implementados.

A biblioteca Jasmine foi a escolhida em detrimento de outras bibliotecas por corresponder às expectativas necessárias para os testes que têm de ser implementados para os componentes, bem como por questões de compatibilidades com as ferramentas de execução dos testes que fazem parte do projeto na fase da sua implementação.

3.3.2 Protractor

O Protractor[24] é uma ferramenta de teste *end-to-end* para aplicações AngularJS. Esta ferramenta executa os testes, implementados em JavaScript, na aplicação em execução, utilizando um ou vários navegadores web reais. O Protractor controla os navegadores e simula as ações de um utilizador da aplicação.

O ficheiro de configuração do Protractor, protractor-conf.js, tem as informações relativas aos *drivers* dos navegadores em que se pretende testar a aplicação, bem como a localização da

[23] http://jasmine.github.io/2.0/introduction.html
[24] http://angular.github.io/protractor/#/

ferramenta Selenium WebDriver[25] na qual assenta a ferramenta Protractor. De notar que a ferramenta Selenium WebDriver foi implementada como um conjunto de extensões, para controlar navegadores.

O Protractor usa o Jasmine para a sintaxe dos testes.

3.3.3 Karma

A ferramenta Karma[26] permite a execução de testes JavaScript em múltiplos navegadores reais. Ao contrário do Protractor, esta ferramenta executa testes de unidade, entre outros. Na realidade, o Karma, somente inicia um servidor HTTP e gera um arquivo HTML de execução de testes.

Na altura de implementação dos testes de unidade, o programador poderá escolher a biblioteca JavaScript que preferir, pois, já existem *plugins* para a maioria das ferramentas de testes: Jasmine, Mocha, QUnit, etc. No projeto, optou-se pela biblioteca Jasmine, por ser compatível tanto pela ferramenta Karma, como pela ferramenta Protractor.

O ficheiro karma.conf.js contém todas as configurações da ferramenta Karma, nomeadamente, o caminho com todos os ficheiros necessários para executar os testes de unidade. O ambiente de desenvolvimento configurado permite desta forma aumentar a produtividade, diminuindo o tempo de espera caraterístico no desenvolvimento de aplicações web entre as modificações feitas e o resultado produzido.

3.4 Estrutura de Diretórios

O código do projeto tem de ser modular e por isso é extremamente importante definir uma boa estrutura de diretórios, de modo a que esta faça sentido para outros programadores que venham a analisar e utilizar o código. A boa organização do projeto facilita a sua manutenção, aumenta a velocidade de desenvolvimento e possibilita identificar problemas antecipadamente. Tratando-se de um projeto implementado com AngularJS este tem de manter a estrutura que é esperada para um projeto deste tipo.

Como referido anteriormente, foi utilizado a ferramenta Yeoman que ajuda nesta tarefa, criando uma estrutura inicial. Contudo, os módulos e configurações obtidos por *default* usando esta ferramenta não são suficientes para o projeto que se pretende implementar. Neste sentido, foi

[25] http://docs.seleniumhq.org/projects/webdriver/
[26] http://karma-runner.github.io/0.12/index.html

necessário instalar mais módulos, complementar as configurações das ferramentas e definir novas tarefas para executar com o Grunt.

Para este projeto foi necessário criar novos diretórios para os testes e para as bibliotecas externas que não foram instaladas no projeto usando nem o npm nem o Bower. Como por exemplo a biblioteca Highcharts.

Por ter um estrutura um pouco alterada daquela que é obtida usando o Yeoman e por ser um projeto que será reutilizado e intregrado futuramente com outros produtos da empresa, decidiu-se fazer uma breve análise da mesma, acompanhando cada linha da estrutura de diretórios uma breve explicação.

A estrutura do projeto é composta pelos seguintes diretórios e ficheiros:

```
app/                    → contém os arquivos de origem para a aplicação
  components/           → contém todos as dependências do projeto de Bower
  lib/                  → contém as bibliotecas externas (nem Bower, nem Node.js)
  scripts/              → contém os ficheiros JS com a lógica comportamental
    nsnCharts/          → contém todos os ficheiros JS dos gráficos
    nsnDataview/        → contém a lógica do componente de vista da informação
    nsnTable/           → contém os ficheiros JS das tabelas
    app.js              → módulo principal da aplicação
    appCharts.js        → módulo com funcionalidades comuns aos gráficos
    services.js         → módulo com todos os serviços necessários para a aplicação
  styles/               → contém os ficheiros CSS, com as regras de estilo
  templates/            → contém todos os templates das diretivas da aplicação
  index.html            → página principal da aplicação
coverage/               → contém os webDrivers para executar os testes e2e
node_modules            → contém todos as dependências do projeto de Node.js
test/                   → contém todos os ficheiros de testes
  e2e/                  → contém todos os testes end-to-end (e2e)
  unit/                 → contém todos os testes de unidade
  karma.conf.js         → ficheiro de configuração do Karma
  protractor-conf.js    → ficheiro de configuração do Protractor
.bowerrc                → ficheiro configuração do Bower
.jshintrc               → ficheiro configuração do JSHint
bower.json              → ficheiro com as dependências Bower
Gruntfile.js            → ficheiro configuração Grunt
package.json            → ficheiro configuração com as dependências Node.js
README.md               → ficheiro ajuda para lançar tarefas do Grunt
```

3.5 Arquitetura e Padrões do Projeto

Agora que todos os requisitos de sistema e todas as ferramentas do ambiente de desenvolvimento foram analisadas, vão agora ser analisados a descrição da arquitetura do projeto bem como a descrição de alguns conceitos e padrões inerentes ao desenvolvimento de componentes web.

Para melhor compreensão dos componentes web decidiu-se desenhar a arquitetura no qual se enquadra o projeto. A Figura 3.4 apresenta a base de funcionamento entre um servidor de *Web Services* e os componentes web implementados, em contrapartida com o modo de funcionamento do projeto NPM onde se pretende integrar os componentes do projeto. Como anteriormente descrito, este servidor de serviços web não está implementado e por isso foi simulado. Os componentes implementados em JS, quando precisarem de dados irão efetuar um pedido HTTP a um servidor web. Este pedido é assíncrono e retorna os dados num ficheiro JSON.

No caso do projeto NPM, este está maioritariamente implementado em Flex e Java e espera que os dados sejam devolvidos no formato XML, tal como definido na Figura 3.4.

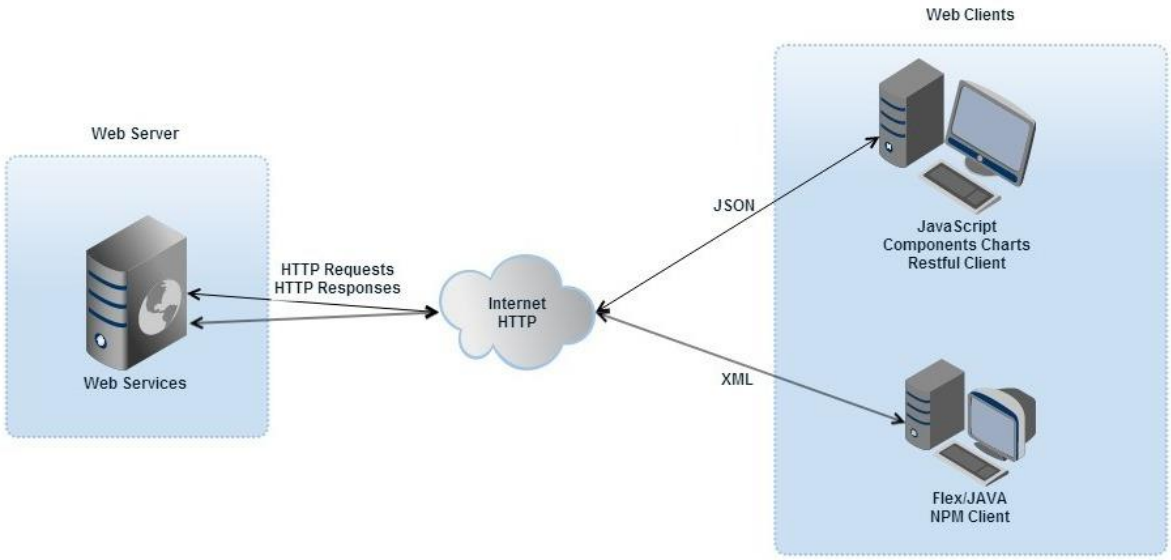

Figura 3.4 – Arquitetura do projeto

Como neste momento, não existe uma *open* API com os serviços web necessários, o NPM utiliza internamente uma API, considerada internamente na empresa, *semi-open* API. Foi considerada *semi-open* API pois alguns serviços são públicos ao passo que outros, usados igualmente pelo produto NPM, são privados. Como os componentes web já estão implementados usando a lógica de que os dados são recebidos através de pedidos HTTP, quando os serviços web forem implementados, a alteração de código necessária é mínima. Utilizando o serviço $resource disponibilizado pelo AngularJS é possível interagir com um servidor RESTful.

Para implementar a solução temporária do servidor web com os dados no formato JSON foi necessário configurar filtros CORS no servidor. O servidor usado é Apache Tomcat 8.0, sendo o conceito CORS o mecanismo que permite que os recursos do servidor, neste caso os ficheiros com os dados JSON, sejam acedidos pela aplicação que contem os componentes com os dados para apresentar. Os filtros CORS são necessários quando servidor e aplicação web estão em

domínios diferentes. É nestes filtros que se podem configurar quais os domínios das páginas web que podem aceder aos recursos e quais as que não têm permissões para tal.

Os ficheiros com os dados que são enviados pelo servidor web têm uma estrutura bem definida, que foi previamente acordada entre os clientes e servidor. Só desta forma é possível analisar e perceber os dados que são recebidos. Um ficheiro de dados com formato JSON enviado pelo servidor web tem tipicamente o seguinte aspeto da Figura 3.5.


```
{
    frozenColumns: 1,
    visibleColumns: 8,
  - columns: [
    - {
          name: "Column A",
          id: "a"
      },
    - {
          name: "Column B",
          id: "b"
      },
    - {
          name: "Column C",
          id: "c"
      },
    - {
          name: "Column D",
          id: "d"
      },
    - {
          name: "Column E",
          id: "e"
      },
    - {
          name: "Column F",
          id: "f"
      },
    - {
          name: "Column G",
          id: "g"
      },
    - {
          name: "Column H",
          id: "h"
      }
  ],
  - data: [
    - {
          a: 23,
          b: 98,
          c: 79,
          d: 57,
          e: 59,
          f: 12,
          g: 32,
          h: 93
      },
    - {
          a: 32,
          b: 89,
```


Figura 3.5 – Exemplo de dados JSON recebidos

De notar que o formato JSON foi criado como um subconjunto da notação de objeto do JavaScript, o que resulta numa incrível compatibilidade entre ambos. No projeto optou-se por usar ferramentas e bibliotecas baseadas em JavaScript, sendo esta linguagem baseada em objetos, a troca de informação é sempre feita no formato JSON.

Uma análise muito importante é a integração dos vários componentes web implementados e como podem ser apresentados e organizados numa aplicação web real. A Figura 3.6 representa uma possível integração dos componentes HTML5 com o produto NPM. Para facilitar a explicação de cada espaço da aplicação foram definidas zonas numéricas e descritas com detalhe a que correspondem cada uma destas zonas.

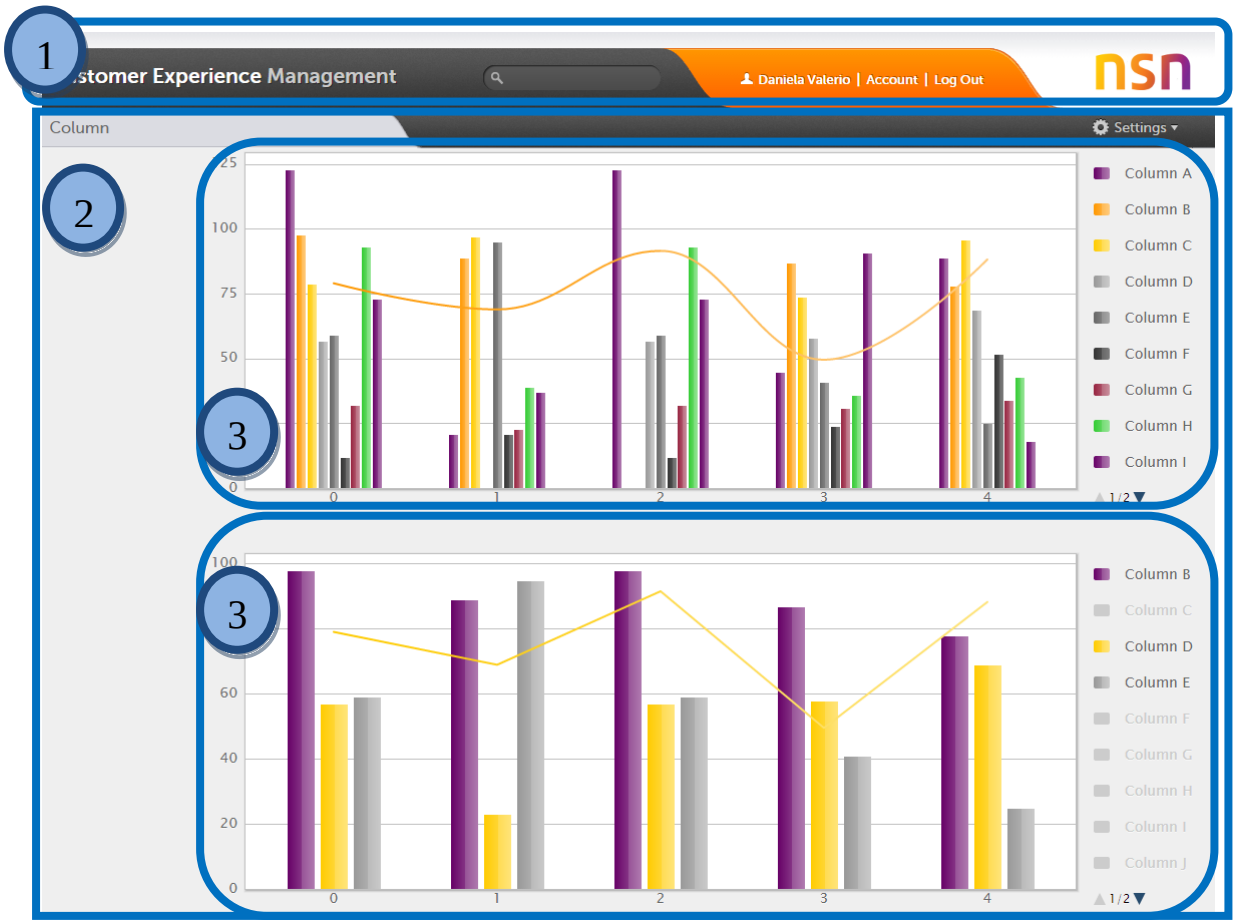

Figura 3.6 – Integração dos diferentes componentes web

1 – Corresponde a uma área inativa da aplicação. Serve para dar uma visão mais realista de como será o resultado da aplicação depois da integração destes componentes no produto NPM da empresa.

2 – Esta zona corresponde ao componente web definido como “Data View”. Este componente é o componente responsável por agregar os outros componentes web, tanto gráficos como tabelas. Nenhum componente web de gráficos ou tabelas pode existir sem estar contido num “Data View”. Este pode conter um ou mais componentes web, que podem interagir entre si. No caso da Figura 3.6, o “Data View” contém dois componentes web de gráficos. Este componente é uma diretiva implementada em AngularJS com nome de elemento HTML:

```
<nsn-dataview></nsn-dataview>
```

3 – Esta zona corresponde a componentes web responsáveis pela apresentação e manipulação dos dados. Estes componentes podem ser tabelas ou gráficos e conseguem interagir entre si. No caso da Figura 3.6 o componente web "Data View" tem integrado dois tipos de gráficos mistos: colunas com *spline* e colunas com linha.

Cada componente corresponde à implementação de uma diretiva do tipo "AE". "A" porque pode receber atributos e "E" porque se comporta como um elemento HTML. Foi implementado uma diretiva por cada tipo de gráfico a representar e também para as tabelas. Por exemplo, para o componente web de gráficos de linhas, está implementado uma diretiva com o nome de elemento HTML: nsnChartLine. De notar que como todas as diretivas foram implementadas sendo "AE", cada diretiva pode receber atributos, sendo estes opcionais.

Um exemplo prático de um componente web na categoria de gráficos de linhas que receba um atributo é possível utilizar implementando a seguinte linha de código HTML:

```
<nsn-chartLine typeChart="spline"></nsn-chartLine>
```

O resultado visual deste componente está na Figura 3.7 que apresenta os dados num gráfico de *spline*, ou seja, com linhas curvilíneas:

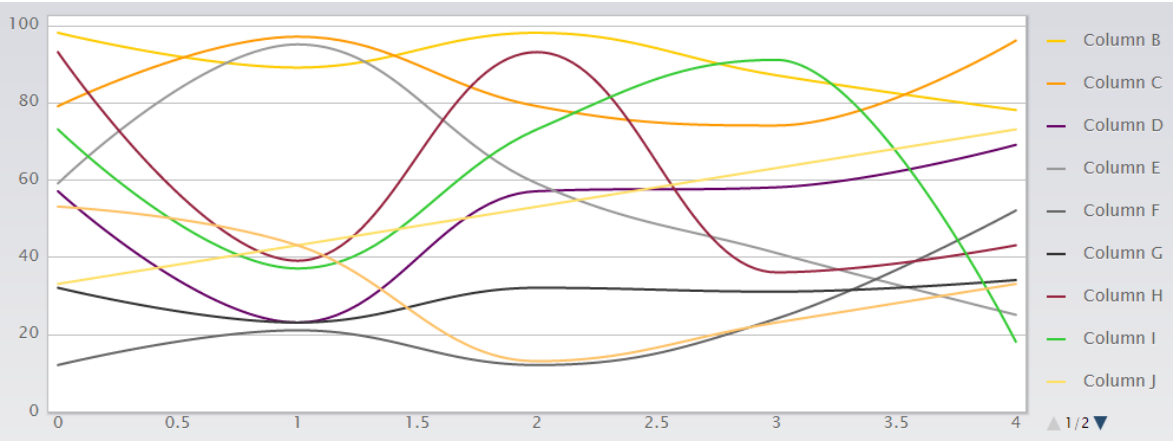

Figura 3.7 – Componente Gráfico de linhas curvilíneo

No futuro é perfeitamente possível implementar mais diretivas, para outros tipos de gráficos, bem como definir novos atributos para as diretivas já implementadas. Contudo, as diretivas implementadas serão abordadas no capítulo da implementação do projeto.

Uma característica interessante e extremamente importante nos componentes criados é a capacidade de poderem interagir entre si. Por exemplo, considerando dois componentes na mesma "Data View", sendo um componente do tipo tabela e o outro componente um gráfico

qualquer é possível adicionar no gráfico valores que sejam selecionados a partir da tabela. Neste contexto que surge o conceito *two-data-binding*, que está relacionado com o padrão de software MVC implementado neste projeto e que é analisado a seguir.

Padrão MVC

MVC é um padrão de software bastante implementado nos dias de hoje no desenvolvimento de aplicações web. Este modelo de arquitetura de software separa a representação da informação, da interação do utilizador. MVC deriva de *model view controller*, em que *model* representa a lógica, funções e os dados da aplicação. *View*, consiste na representação da informação que pode ser de várias formas: uma tabela, diagrama ou gráfico. A vista é o componente que solicita do modelo a informação que ela necessita para gerar uma representação de saída. *Controller* advém da capacidade de aplicar níveis de interatividade entre o *model* e a *view,* este serve de mediador. O controlador envia comandos para as respetivas vistas a que está associado e que pretende atualizar. Também é o controlador que envia comandos para o modelo de forma a atualizar o seu estado. O modelo notifica as vistas e os controladores associados quando há uma mudança do seu estado. É com base nesta notificação que as vistas são atualizadas e que os controladores alteram o conjunto de comandos disponíveis. Este modo de funcionamento é apresentado de forma clara na Figura 3.8.

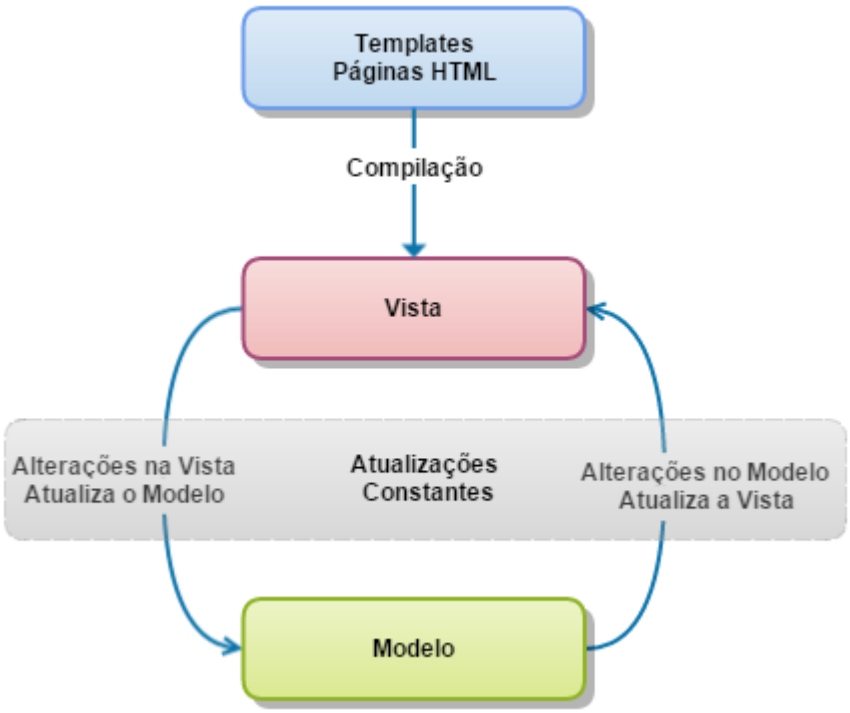

Figura 3.8 – Diagrama de interação MVC

O padrão MVC foi descrito pela primeira vez em 1979 por Trygve Reenskaug. No artigo de Steve Burbeck, Ph.D. com o título: “Applications Programming in Smalltalk-80: How to use Model–View–Controller” é feita uma descrição profunda sobre a sua implementação original. Apesar de desenvolvido originalmente para computação pessoal, o MVC foi amplamente adaptado como uma arquitetura para aplicações web das linguagens de programação mais conhecidas. Este tipo de arquitetura tem de dividir e decidir as responsabilidades entre o cliente e servidor. Umas ferramentas preferem colocar maior responsabilidade do lado do servidor deixando assim o cliente mais liberto, ou passo que outras preferem evitar ao máximo os pedidos do cliente ao servidor, o que permite que os componentes MVC sejam executados parcialmente no cliente, através de pedidos AJAX.

Este padrão além de dividir a aplicação em três tipos distintos também define as interações entre eles. As principais vantagens do MVC são a reusabilidade de código e separação de conceitos. Com o aumento da complexidade das aplicações desenvolvidas, sempre visando a programação orientada a objeto, torna-se relevante a existência de uma clara separação entre os dados e a apresentação das aplicações. Desta forma, alterações feitas nas vistas não afetam a manipulação de dados e vice-versa, estes poderão ser reorganizados sem alterar as vistas.

No caso prático de uma aplicação web, a visão corresponde ao documento HTML. O controlador recebe uma entrada GET ou POST após um estímulo do utilizador e decide como processá-la, invocando objetos do domínio para tratar a lógica de negócio. Por fim, consoante as ações do utilizador é invocado uma vista para apresentar a saída, ou seja, o resultado obtido depois de detetadas as alterações do utilizador feitas na aplicação.

A ferramenta AngularJS, usada para implementação do projeto é conhecida por implementar um padrão de software do tipo MV* ou MVW, com W para *Whatever* no sentido de: "whatever works for you” [10]. No capítulo 2, Tecnologias Web, onde são analisadas várias ferramentas web, a ferramenta AngularJS é uma das que implementa o conceito *two-data-binding*, que está relacionado com o padrão MVC. No AngularJS é possível tirar partido deste conceito usado o “$watch”, que consiste num *listener* específico que despoleta uma ou várias ações a pedido do programador caso seja detetada uma alteração no valor ao qual se está a efetuar o $watch. Um exemplo prático deste conceito no projeto é a implementação de alarmes, com base em valores de *threshold* máximos definidos pelo utilizador em tempo real. À medida que o utilizador vai alterando o valor de *threshold* a aplicação vai detetar esta mudança e atualizar o número de alarmes que deve despoletar no gráfico. Estes alarmes correspondem aos círculos a vermelho no gráfico, que, na realidade, são marcadores que podem ser redefinidos para outro qualquer formato ou cor.

Como se pode ver na Figura 3.9, só três valores no gráfico violam o valor de *threshold* máximo definido e por esse motivo só estão desenhados 3 círculos a vermelho.

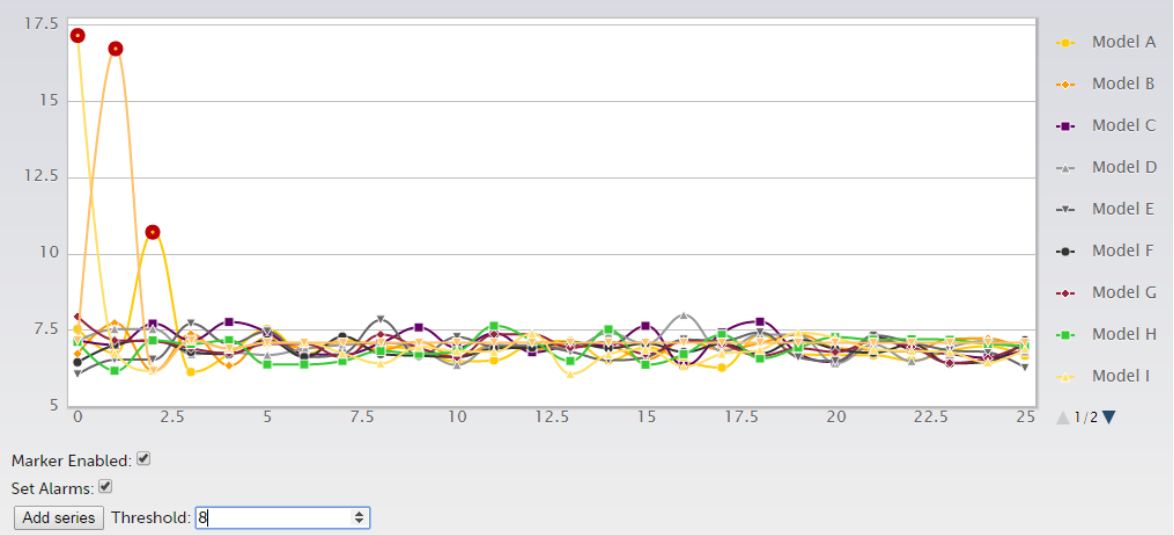

Figura 3.9 – Exemplo prático do padrão MVC

Com a alteração do valor máximo do *threshold*, a deteção de mudança de valor é automática e a vista para o utilizador é atualizada. O novo gráfico tem o aspeto da Figura 3.10.

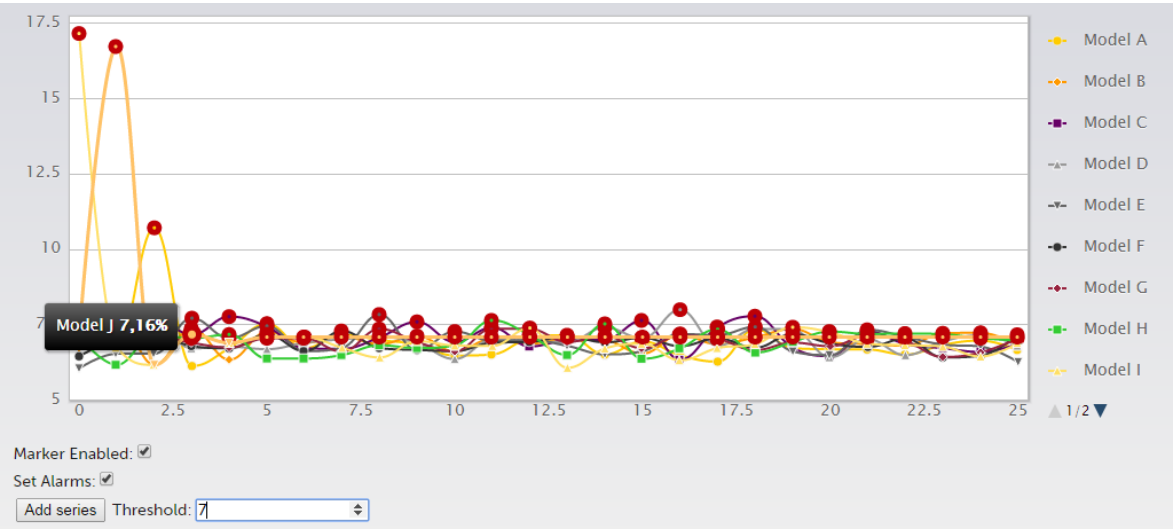

Figura 3.10 – Atualização do valor máximo de *threshold*

Utilizando o padrão MVC presente na ferramenta AngularJS, só com um click, o utilizador consegue facilmente manipular os dados do gráfico, aumentando e facilitando a sua interação. Desta forma, o utilizador consegue, com menor esforço, tirar maior partido do significado dos dados e facilmente tirar conclusões importantes em relação aos mesmos.

3.6 Síntese

Neste capítulo foram descritos todos os requisitos e toda a arquitetura onde de enquadra o projeto. Foi também apresentado um exemplo com a integração dos diferentes componentes web que compõem uma possível vista de uma situação real do produto NPM. São também apresentadas as ferramentas que acompanharam toda a fase de implementação do projeto, tanto ferramentas para desenvolvimento, como por exemplo o IDE escolhido, como as ferramentas necessárias para a implementação e execução dos diferentes tipos de testes: *end-to-end* e de unidade.

Juntamente com a análise da arquitetura foi abordado um padrão de software muito importante, o MVC, bem como a forma como os componentes podem interagir entre si. Sendo um projeto que será reutilizado e possivelmente complementado com mais componentes web no futuro, foi feita uma breve explicação da estrutura de diretórios que compõem o projeto.

Este capítulo é muito importante pois permite compreender como são recebidos os dados a apresentar nos componentes e compreender que os componentes implementados recebem os dados em formato JSON, o que só por si trará maior desempenho no produto final.

4 Implementação dos componentes web

Neste capítulo é abordada toda a implementação inerente aos componentes web, desde os serviços e *factories* internos necessários para os componentes, bem como as diretivas implementas que são por si os componentes web. Vão também ser analisados padrões de design UI, nomeadamente a implementação das *media queries*. O TDD por ser uma metodologia que acompanhou a fase de implementação será também neste capítulo analisado e dividido entre testes de unidade e testes *end-to-end*.

Por fim, foram implementadas funcionalidades que não estavam previstas no início do projeto mas que são uma mais-valia e por isso, também serão abordadas e retiradas algumas conclusões sobre as mesmas, nomeadamente a atualização do *look&feel* para OT3 (Ocean Touch) e implementação deste projeto, contendo os componentes web desenvolvidos, numa aplicação web móvel capaz de executar em dispositivos android.

4.1 Serviços

Para implementação de determinadas funcionalidades recorrentes e necessárias pela maioria dos módulos e respetivas diretivas decidiu-se implementar um módulo denominado por "Services". Neste módulo estão implementas várias *factories*, como normalmente são nominadas em AngularJS. Estas *factories* consistem num conjunto de funções, que são considerados os serviços disponibilizados por cada *factory* e que fazem um pré-processamento ou processamento intermédio e devolvem o resultado desse processamento ao módulo que chamou pelo serviço pretendido.

Neste módulo, foram implementas duas *factories,* a "infoFactory" e a "createChartFactory". A "infoFactory" é responsável pelos dados: tem implementadas funções para efetuar os pedidos HTTP para obter os dados JSON a apresentar e faz o processamento dos dados recebidos em JSON. Este processamento seve para que os dados sejam agrupados por series e por categorias de forma a poderem ser utilizados nas tabelas e nos diferentes tipos de gráficos.

As *factories* são os serviços que, computacionalmente, requerem o maior esforço de todo o projeto. São os serviços que implementam grande parte da lógica comportamental, sendo as diretivas os clientes que precisam e irão utilizar estes serviços. Desta forma, o código das diretivas é mais fácil de compreender e o projeto, no seu conjunto, bem definido e estruturado por módulos de AngularJS.

Um exemplo de trecho de código do módulo "infoFactory" é apresentado de seguida:

```
(function () {
    "use strict";
    angular.module("services", [])

        .factory('infoFactory', ['$http', function ($http) {
            return {
            formatCategories: function (results) {
                      var maxRows = results.data.length;
                      var id = [], categories = [];

                      for (var i = 0; i < maxRows; i++) {
                          if (typeof results.columns[0] !== 'undefined') {
                              id[i] = results.columns[0].id;
                          }

                          categories[i]= results.data[i][id[i]];
                      }
                      return categories;
                  },
              (…)
```

O exemplo apresentado contém a função, ou serviço, que permite agregar os dados recebidos do ficheiro JSON e agrega-los em categorias para que sejam corretamente apresentados nos gráficos e tabelas. Esta função é deveras a mais simples e por esse motivo o exemplo utilizado.

Outro aspeto importante e visível no código apresentado está nas dependências que precisa. O serviço $http é do AngularJS e permite a comunicação com servidores através do protocolo HTTP. Neste exemplo é visível a injeção de dependências. Outra consideração tida em conta na fase de implementação foi a forma como é escrito o código do projeto, esta é feita de forma a simplificar a posterior minificação dos ficheiros.

A outra *factory* implementada tem o nome “createChartFactory” e como o nome indica, contem todos os serviços para a criação dos diferentes tipos de gráficos. Esta *factory,* ao contrário da analisada anteriormente não requer nenhum tipo de injeção de dependências de módulos ou serviços.

4.2 Diretivas

A biblioteca AngularJS permite a implementação das nossas próprias diretivas. As diretivas são elementos de HTML personalizados que permitem criar as funcionalidades pretendidas. Estes elementos estão encapsulados, são interoperáveis e reutilizáveis. É com base nas diretivas do AngularJS e nos seus conceitos que são implementados todos os componentes web deste projeto.

A representação da informação é feita com uso de tabelas e diferentes tipos de gráficos. Para cada tipo de representação foi implementado uma diretiva correspondente. Cada diretiva tem

associado um *template* com o HTML que é gerado para a respetiva vista, desta forma, caso se pretenda posteriormente alterar a vista, basta alterar o código do *template*.

Como diretiva inicial, foi implementado o componente que irá conter todos os outros componentes. Como descrito anteriormente a este capítulo, falamos da diretiva “Data View” que permite conter um ou vários tipos de componentes de tabelas e/ou gráficos. Para utilizar esta diretiva basta o programador escrever a seguinte linha de código:

```
<nsn-dataview> </nsn-dataview>
```

A implementação desta diretiva, por ser tão simples e não ter lógica comportamental associada é um módulo muito simples:

```
(function () {
    "use strict";

angular.module("nsnDataview", [])
    .directive('nsnDataview', function () {
        return {
            transclude: true,
            restrict: 'E',
            scope: {},
            templateUrl: 'templates/dataView.html',
            replace: true,
            link: function linkFn(scope) {}
        };
    });
}());
```

Para cada diretiva foi implementado o *template* html correspondente. No caso desta diretiva foi implementado o ficheiro dataView.htlm, também ele muito simples:

```
 <div class="dataView" ng-transclude>
    <div id="myDIV">
    </div>
</div>
```

O *template* da diretiva “Data View” implementada tem de ter a diretiva “ng-transclude” do AngularJS, bem como a propriedade “transclude: true” na diretiva. Somente desta forma é indicado ao AngularJS que este é um ponto de inserção para DOM. Isto significa, que o conteúdo existente dentro do elemento desta diretiva será introduzido como parte desta diretiva. Como esta diretiva contém outros componentes, esta propriedade é extremamente importante e sem ela, nenhum componente integrado com esta diretiva seria visível.

Para integrar nesta diretiva outras diretivas é necessário implementar, por exemplo as seguintes linhas de código html:

```
<nsn-dataview id="dataView">
    <nsn-table filejson="input.json"></nsn-table>
    <nsn-columnchart filejson="input.json"></nsn-columnchart>
</nsn-dataview>
```

A utilização dos componentes anteriores resultaria numa vista composta por três componentes web, o “Data View” que contém os componentes, um gráfico de colunas com os dados do ficheiro “input.json” e uma tabela com esses mesmos dados e ou seja, uma vista equivalente à Figura 4.1.

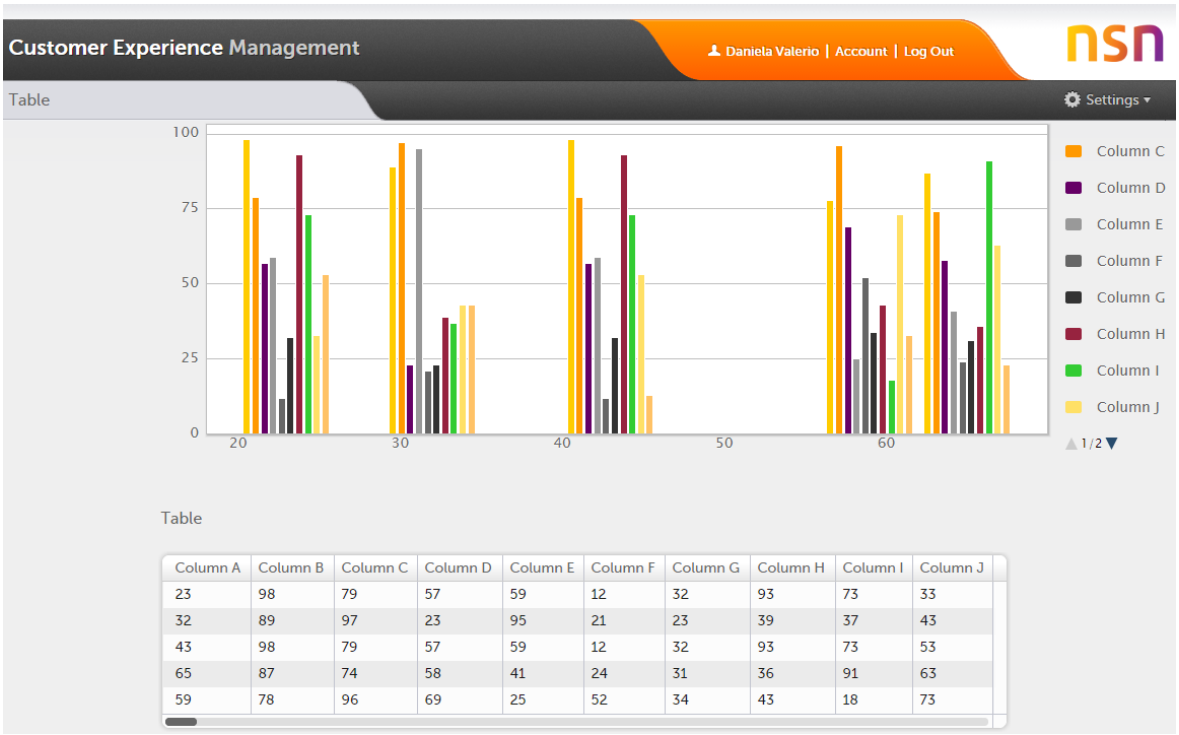

Figura 4.1 – Exemplo de integração de diferentes componentes

Para além da implementação da diretiva “Data View” foram implementas outras diretivas, algumas já anunciadas anteriormente. A implementação das diretivas foi divida em dois grupos bem distintos, as diretivas para tabelas e diretivas para os gráficos. Cada um destes grupos é descrito nas secções seguintes.

4.2.1 Tabelas

A representação de informação usando tabelas é muito comum neste género de aplicações mas este tipo de vista não é muito interativo e por isso é muitas vezes complementado com outros tipos de vistas, como por exemplo, um gráfico.

A vista apresentada na Figura 4.2 corresponde à vista de dois componentes de tabelas, definidas com o mesmo tipo de diretiva, a diretiva “nsn-table”. Cada componente recebe os mesmos dados com a única diferença que uma tabela tem filtros e a outra não tem.

A diretiva “nsn-table” é implementada num módulo AngularJS, ao qual se designou de “nsnTable”. Este módulo recebe como dependência o módulo “services” que como o próprio nome indica, disponibiliza serviços. Como anteriormente explicado é no módulo dos serviços que está implementada toda a lógica dos pedidos de HTTP ao servidor para obtenção dos dados.

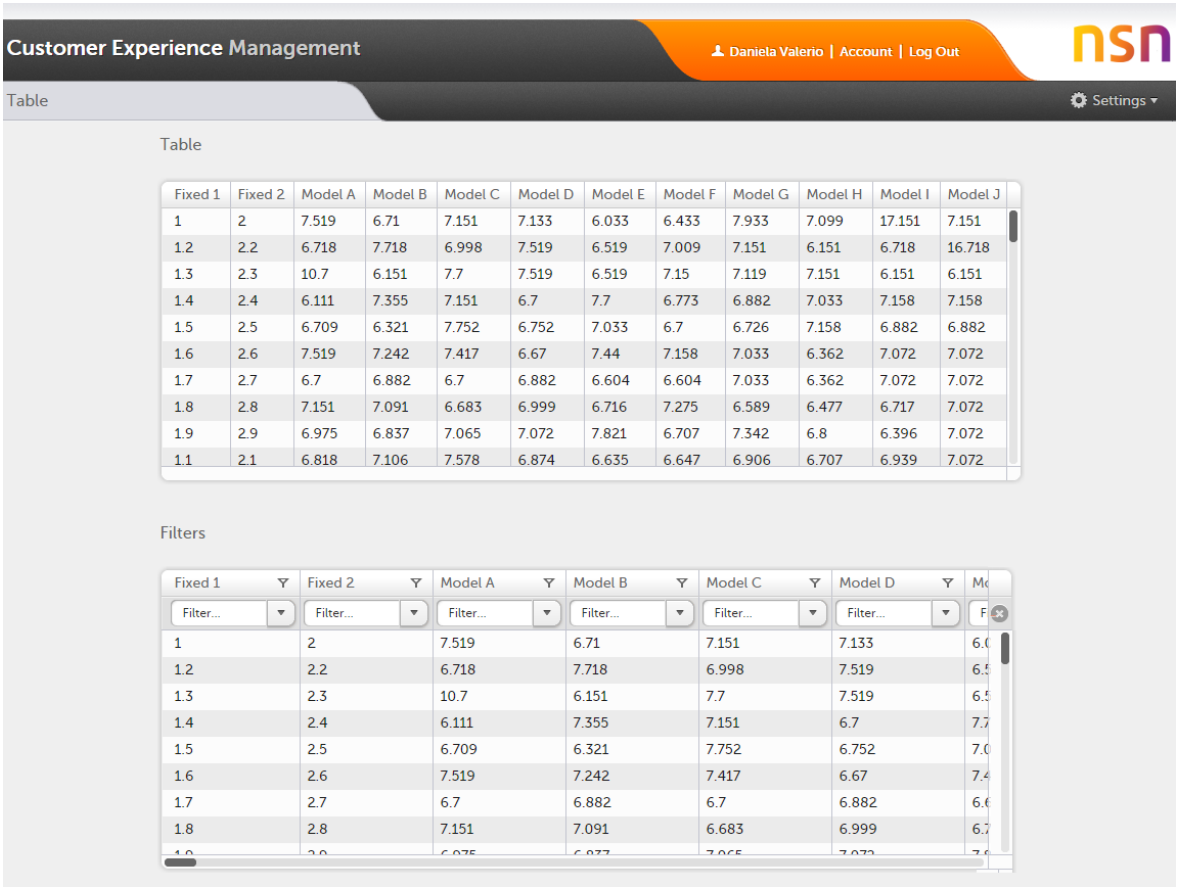

Figura 4.2 – Exemplos de componentes de tabelas

Nas diretivas AngularJS existem parâmetros que podem ser especificados. Como por exemplo, o tipo de diretiva que se pretende implementar. Este parâmetro tem o nome “restrict” e pode receber os seguintes valores: ‘A’, ‘E’ ou ‘C’, ou a combinação destes valores, como por exemplo ‘AEC’. ‘A’ – somente corresponde ao nome de um atributo, ‘E’ – somente corresponde ao nome de um elemento, ‘C’ – somente corresponde ao nome de uma classe de CSS. No caso de ser ‘AEC’ pode corresponder a qualquer um dos três, ou seja, ou ao nome de um atributo, ou ao nome de um elemento ou ao nome de uma classe. Esta diretiva e todas as diretivas dos gráficos são do tipo ‘AE’. A diretiva da tabela pode ser usada da seguinte forma:

```
<nsn-table filejson=”JSON/input.json”></nsn-table>
```

Tanto esta diretiva como as diretivas dos gráficos podem receber um atributo “filejson”. Neste atributo é possível indicar o nome do ficheiro que tem os dados que se pretende apresentar. Este atributo pode ser um URL com o caminho completo para o servidor que tem os dados. Internamente este valor está definido num *scope* isolado, ou seja, esta variável só é visível dentro desta diretiva de forma a evitar conflitos e comportamentos inesperados. Para tal foi necessário definir o atributo “filejson” dentro do parâmetro “scope” da diretiva. Como se pretende que este atributo seja considerado uma string, associou-se a este atributo o símbolo ‘@’. Neste caso só se tem vínculo local da propriedade. No caso de se pretender uma vinculação bi-direcional entre a propriedade local e a propriedade “pai” usa-se o símbolo ‘=’. Isto significa que as mudanças efetuadas na propriedade “pai” sejam refletidas na propriedade local. Outro tipo de vínculo que pode ser definido é o vínculo de execução “pai”. Tal significa que é possível executar uma função que esteja no contexto do *scope* “pai”. Na realidade, quando é usado o símbolo ‘&’ é criado um invólucro de uma função e esta aponta para a função “pai”.

As observações feitas anteriormente são visíveis neste bloco de código que somente apresenta as partes interessantes que se pretende analisar.

```
.directive('nsnTable', function () {
   return {
      restrict: 'AE',
      templateUrl: 'templates/table.html',
      scope:
        {
           filejson: "@"
        },
      replace: true,
      controller: ['$scope','infoFactory', function(scope,dataFactory){
              (…)
      }],
      link: function linkFn(scope) {
        (…)
      }
   };
});
```

Todas as diretivas implementadas no projeto recebem o parâmetro “templateUrl”, que também está definido no código anterior. Esta opção do AngularJS permite definir o caminho do ficheiro com o *template* HTML que se pretende. No caso da diretiva das tabelas foi implementado um ficheiro com o nome “table.html”. Neste ficheiro são recorrentemente usadas diretivas disponibilizadas pelo próprio AngularJS. Estas diretivas permitem simplificar funcionalidades recorrentes e evitar redundância de linhas de código HTLM. Por exemplo, com o uso da diretiva “ng-repeat” é possível preencher todas as linhas e colunas da tabela. Esta diretiva percorre automaticamente todos os valores de um determinado objeto. Na realidade faz o clone e repete o elemento DOM várias vezes, mas tudo de forma invisível para o programador, evitando assim que esse trabalho tenha de ser feito por ele. Múltiplas linhas de código ficam visíveis para o

programador como sendo somente uma. Outra diretiva muito usada é a “ng-bind”. Esta diretiva serve para que o AngularJS substitua o conteúdo de texto do elemento HTML pelo valor específico de uma determinada expressão e mais importante, para atualizar o conteúdo do texto quando o valor dessa expressão se alterar. Em vez do uso específico desta diretiva pode-se usar ‘{{‘ ‘}}’ para delimitar a expressão que se pretende vincular. Contudo, estas podem ficar temporariamente visíveis até que a compilação do *template* pelo AngularJS não seja efetuada na totalidade.

Os valores da tabela podem ser ordenados de forma ascendente ou descendente para qualquer coluna da tabela, por esse motivo, usou-se a diretiva “ng-click”. Desta forma, depois de detetado um click em determinada coluna é feito a reordenação dos valores. O AngularJS permite o uso de filtros, sendo um deles o “orderBy”, usando esta diretiva é possível definir quais os valores a reordenar e a ordem pretendida.

Nas regras de estilo definidas no Orange Touch, as tabelas podem ter associados filtros próprios relativos a cada coluna da tabela. A ideia passa por apresentar na tabela somente os valores daquela coluna que respeitem a regra imposta pelo filtro escolhido pelo utilizador. Todos os restantes valores serão omissos. A Figura 4.3 tem um exemplo de escolha de um filtro para aplicar na tabela.

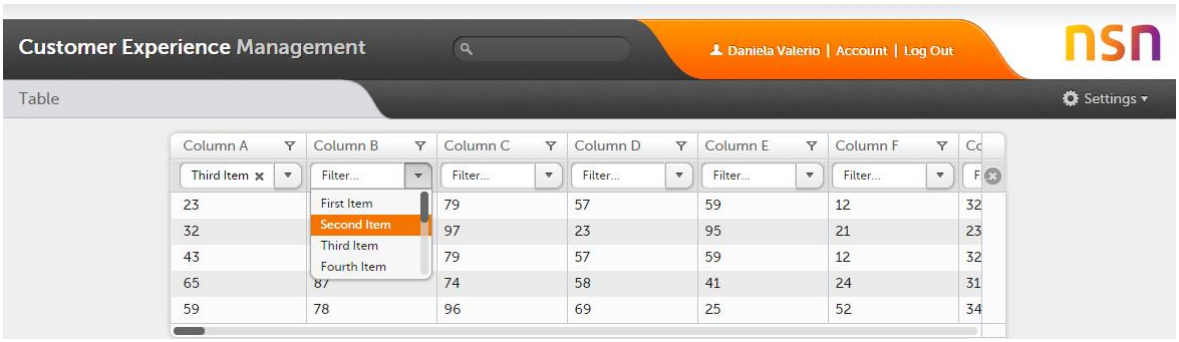

Figura 4.3 – Tabela com filtros

Ainda dentro do contexto de tabelas, decidiu-se implementar outra diretiva com o nome “nsn-selectable”. Esta diretiva difere da anterior porque permite selecionar uma coluna da tabela e os valores dessa coluna serem adicionados automaticamente num gráfico. Esta diretiva e respetiva funcionalidade podem ser observadas na Figura 4.4.

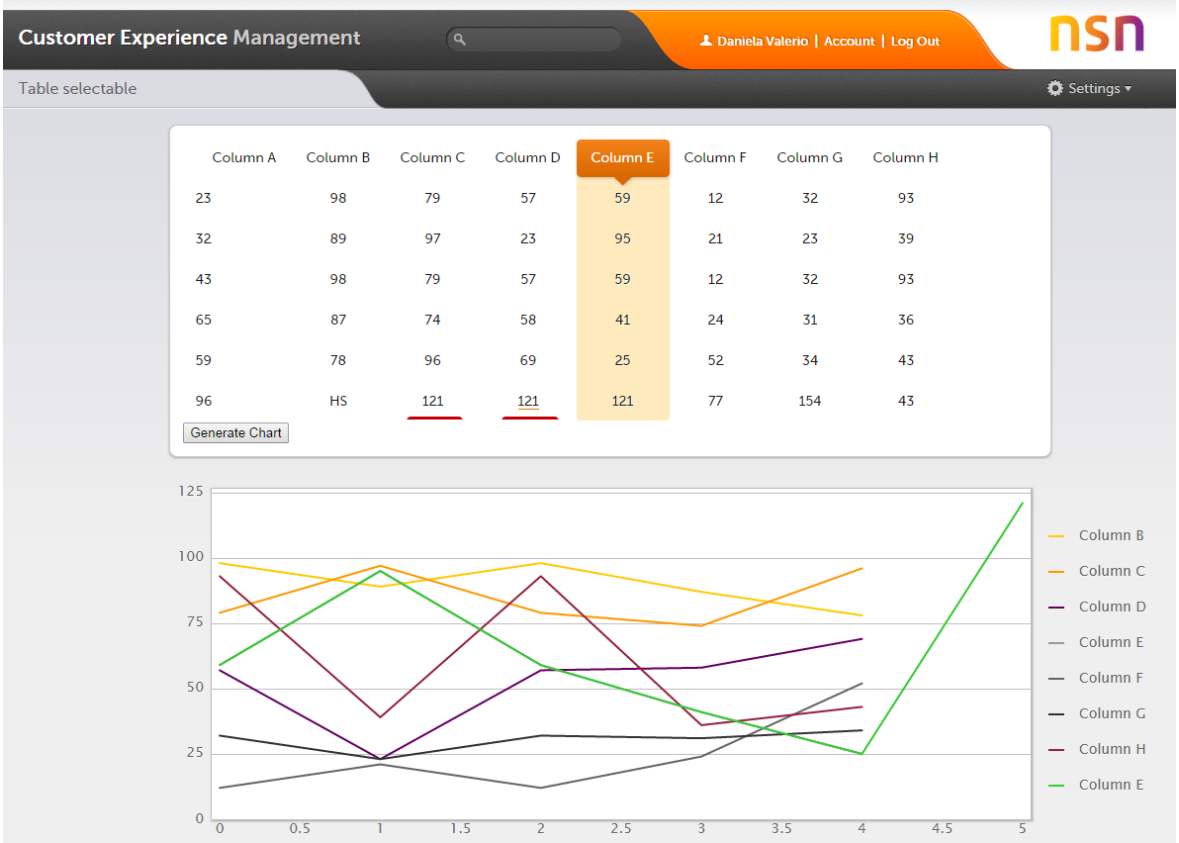

Figura 4.4 – Interação entre tabelas e gráficos

Com este tipo de tabelas selecionáveis é percetível a interação existente entre diretivas diferentes. Esta funcionalidade permite a rápida comparação de valores, facilitando assim a compreensão dos mesmos ao utilizador.

Existem várias diferenças entre os dois tipos de diretivas definidos para as tabelas. Por exemplo, as regras de estilo diferem muito. Por exemplo, nas tabelas selecionáveis, facilmente reparamos que dois dos valores da tabela da Figura 4.4 estão sublinhados a vermelho. Este tipo de formato serve como alerta para valores fora do normal, chamando assim a atenção do utilizador para o problema. Mas ao contrário do outro tipo de diretivas, estas tabelas selecionáveis não permitem a ordenação de valores e não têm filtros associados.

4.2.2 Gráficos

Para esta categoria foram definidos vários tipos de diretivas:

- Linhas
- Barras
- Área
- Tarte

- Colunas
- Colunas Extensíveis

No entanto, para determinados tipos de diretivas é possível a definição de atributos que irão alterar ligeiramente o tipo de gráfico a representar. Utilizando o atributo “typeChart” pode-se especificar o tipo do gráfico a representar.

Os tipos de diretivas que permitem este tipo de atributo são:

- Linhas
- Área

O atributo “typeChart” é facultativo mas quando definido como *spline* em diretivas dos gráficos de linhas ou definido como *areaspline* nos gráficos de áreas, estes ficam com linhas com um aspeto curvo.

Um exemplo prático na categoria de gráficos de linhas seria:

```
<nsn-chartLine typeChart=”spline”></nsn-chartLine>
```

Esta diretiva apresenta os dados num gráfico de *spline*, ou seja, com linhas curvilíneas como se pode observar na Figura 4.5.

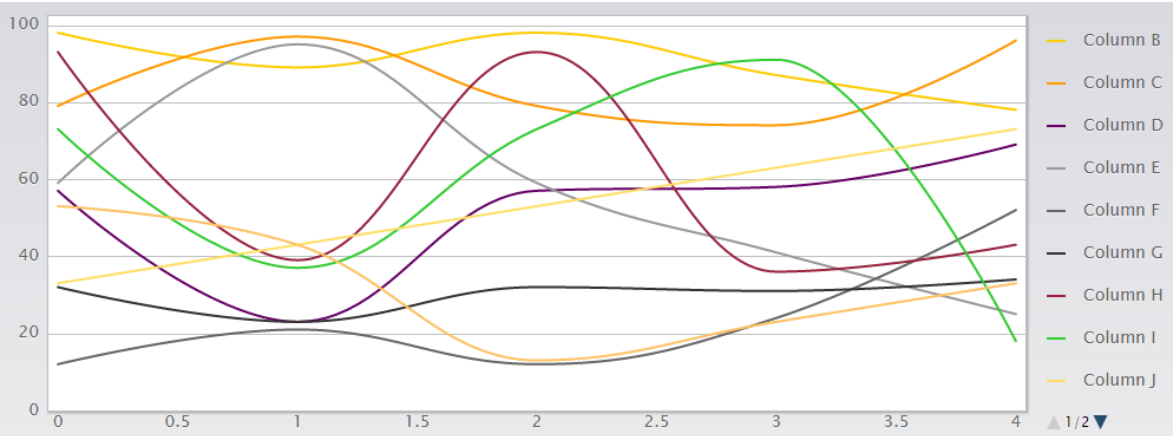

Figura 4.5 – Gráfico de linhas curvilíneo

Outro exemplo de utilização da mesma diretiva mas sem atributo seria:

```
<nsn-chartLine></nsn-chartLine>
```

Nesta diretiva, como o atributo “typeChart” não foi definido, a aplicação representa os dados num gráfico de linhas. Este é o tipo de gráfico considerado como o *default*, dentro da categoria de gráficos de linhas. Ao contrário do gráfico anterior, este corresponde a um gráfico com linhas retas, como é visível na Figura 4.6.

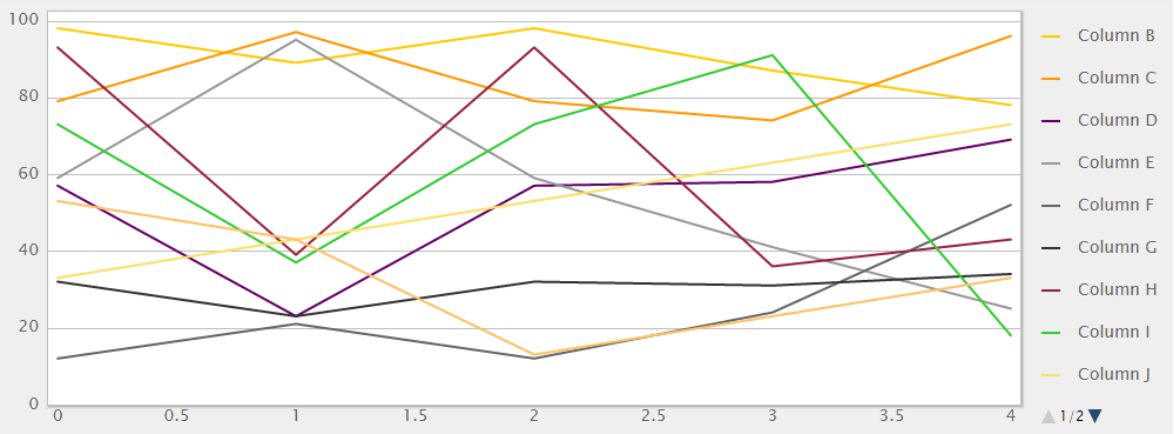

Figura 4.6 – Gráfico linhas retas

Os tipos de gráficos considerados default foram definidos por mim, usando senso comum. Esta nomenclatura foi implementada para juntar na mesma diretiva tipos de gráficos semelhantes e que por isso não precisam da implementação de uma diretiva própria. Consegue-se assim evitar repetição de código usufruindo da reutilização da mesma diretiva.

Por outro lado, utilizando o atributo “combined” específico das diretivas de colunas pode-se criar gráficos mistos:

- Gráficos de colunas com uma série em *spline*, Figura 4.7:

```
<nsn-columnchart filejson="input.json" combined='sline'></nsn-columnchart>
```

- Gráficos de colunas com uma série em linha, Figura 4.8

```
<nsn-columnchart filejson="input.json" combined='line'></nsn-columnchart>
```

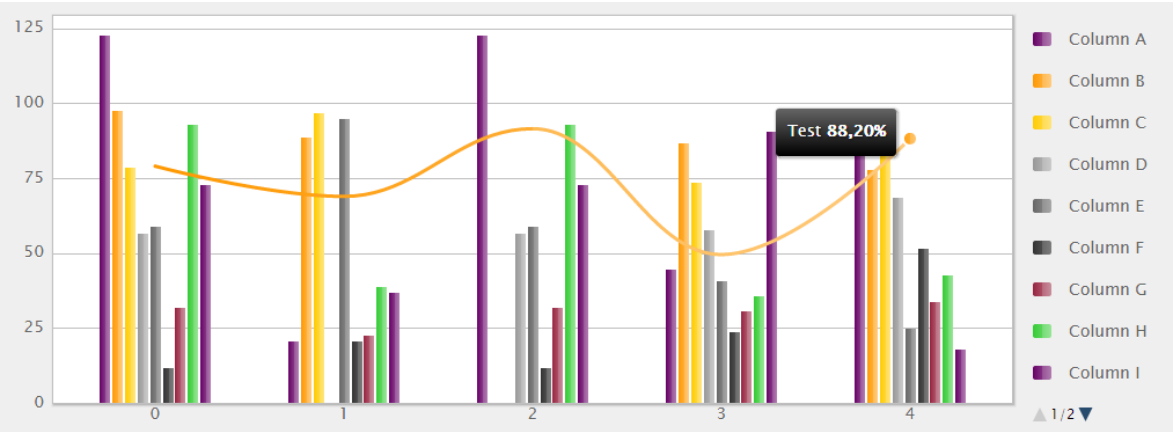

Figura 4.7 – Exemplo gráfico composto: colunas com *spline*

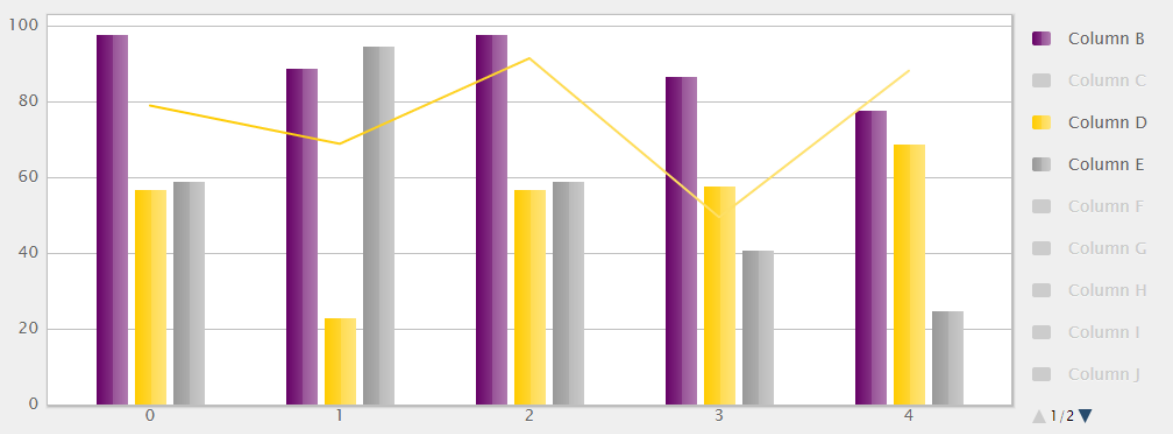

Figura 4.8 – Exemplo gráfico composto: colunas com linha

Os gráficos mistos ou compostos podem ser importantes para questões de comparação de tecnologias ou afins.

Uma funcionalidade implementada que é possível observar na Figura 4.8 consiste na possibilidade de esconder valores do gráfico bastando clicar nas respetivas séries das legendas que se encontram à direita do gráfico e que se pretendem ocultar. Ao clicar numa série da legenda, os valores correspondentes deixam de ser representados no gráfico e este é redesenhado mas com novos valores nos eixos, de forma a representar da melhor forma os valores que são pretendidos pelo utilizador. Para voltar a visualizar os valores, basta clicar de novo na série pretendida que está dentro da legenda. Todas as séries que estiverem desativas terão uma cor cinza claro, quase que ilegíveis nas legendas do gráfico.

No futuro é perfeitamente possível e fácil de implementar mais diretivas, para outros tipos de gráficos, bem como definir novos atributos para as diretivas já implementadas. Por questões de encapsulamento e segurança do código, os atributos estão todos definidos como *isolate scope*.

Foram implementados diversos tipos de gráficos de forma a ampliar o leque de escolha do utilizador. Para cada gráfico existem associadas algumas funcionalidades implementadas que só fazem sentido para determinado tipo de gráfico.

Outro atributo definido nas diretivas é o atributo “filejson”. Este atributo serve para facultar a localização dos dados, quer seja remota ou local.

No caso de gráficos de áreas ou areaspline, estes têm de ter transparência de forma a não se sobreporem os valores a apresentar, Figura 4.9.

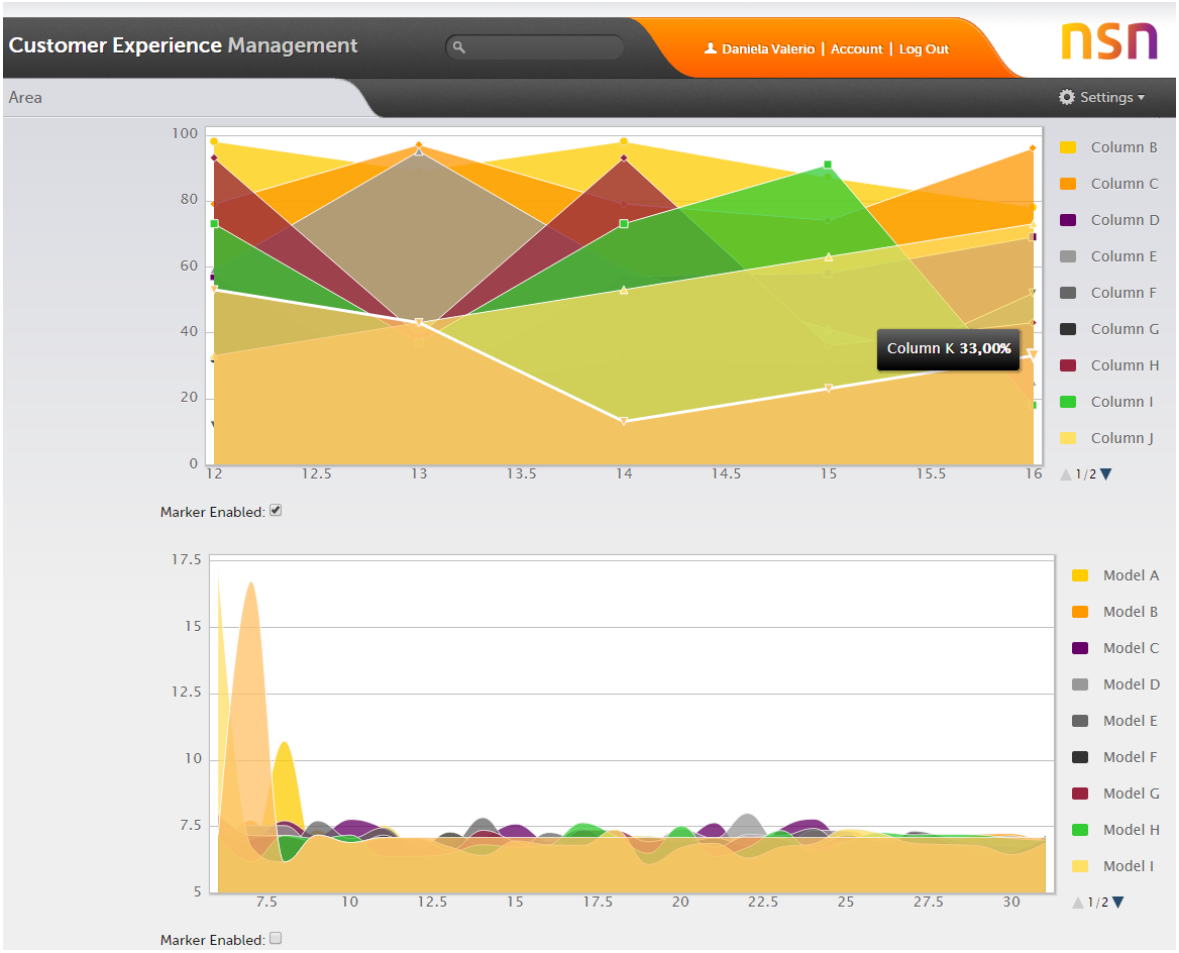

Figura 4.9 – Gráficos de área e areaspline

- Diretiva para gráficos de áreas:

```
<nsn-areachart filejson="input.json"></nsn-areachart>
```

- Diretiva para gráficos de área *spline*:

```
<nsn-areachart filejson="tests.json" typechart="areaspline"></nsn-areachart>
```

Os gráficos pretendem ajudar da melhor forma o utilizador a compreender o estado da sua rede. Para isso é necessário implementar funcionalidades interativas entre os dados e o utilizador de forma a ajudar na rápida compreensão destes dados. Uma das funcionalidades implementadas com esse objetivo passa na possibilidade de o utilizador poder efetuar *zoom* de áreas no gráfico que sejam do seu interesse, por exemplo em picos de valores, como é feito na Figura 4.10 e apresentado o resultado do respetivo *zoom* na Figura 4.11.

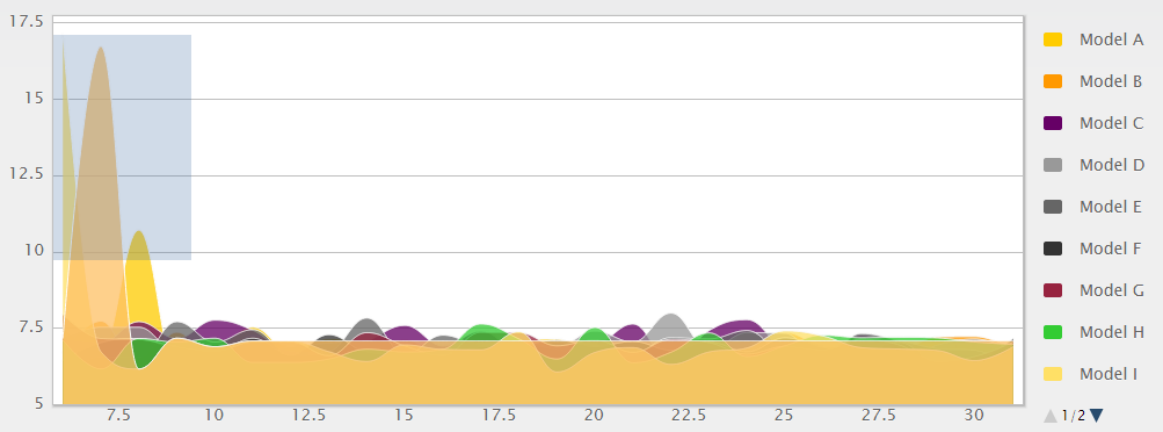

Figura 4.10 – *Zoom* nos eixos YX

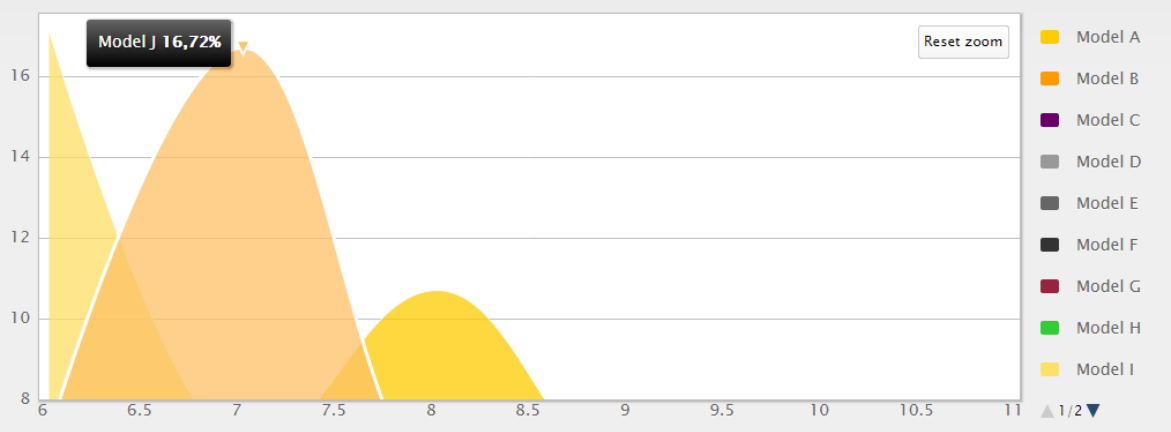

Figura 4.11 – Resultado do *zoom* escolhido

Este tipo de *zoom* é feito com base na área selecionada e considerando ambos os eixos y e x. Mas caso seja preferível, a funcionalidade *zoom* pode ser facilmente alterada e modo a só ser possível fazer *zoom* no eixo y mas mantendo os eixos normais de x do gráfico, ou vice-versa. Esta funcionalidade permite um elevado nível de interação com o gráfico de modo a obter um maior detalhe dos valores apresentados. Para retomar a vista normal do gráfico basta o utilizador clicar no botão “Reset zoom” situado no canto superior direito do gráfico.

Vamos agora analisar a diretiva que implementa gráficos de colunas extensíveis. Supondo conjuntos de dados agregados em diferentes níveis, existindo um conjunto considerado primeiro nível que por sua vez, contém outro conjunto de dados considerado num segundo nível. Para aceder aos dados do segundo nível é necessário efetuar um *drill down* ou extensão dos dados do primeiro nível. Esta funcionalidade implementada no projeto permite agregar no mesmo gráfico muito mais informação que os outros tipos de gráficos.

Considerando a Figura 4.12, vamos por exemplo supor que o utilizador pretende aceder aos dados da coluna H. Depois do utilizador clicar na coluna “Column H” será redesenhado um gráfico de colunas com os dados da tecnologia: “Column H”, como apresentado na Figura 4.13.

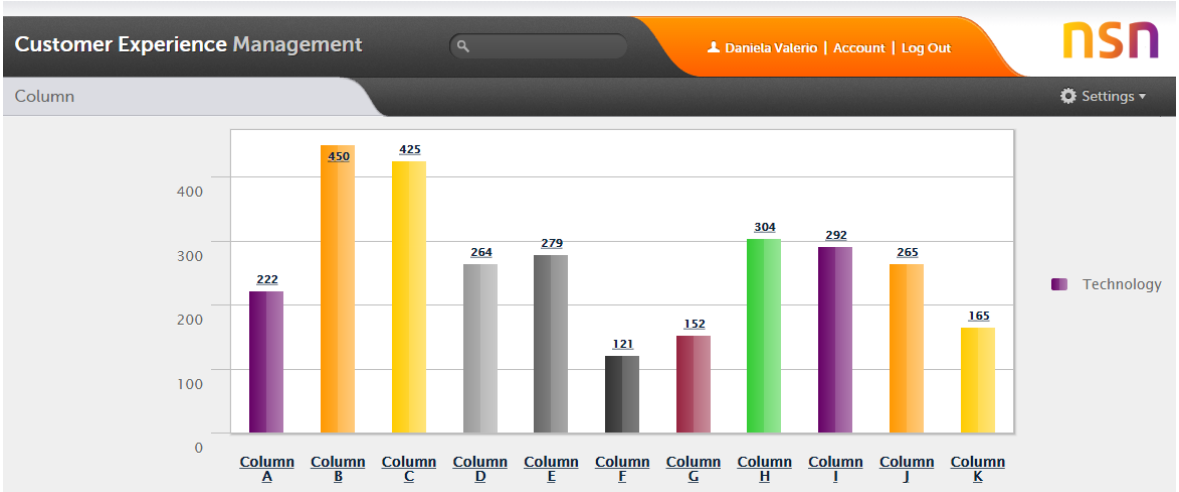

Figura 4.12 – Gráfico com *drill down*

Esta funcionalidade é muito importante pois permite aprofundar a vista do utilizador, entrando em diferentes níveis de apresentação de informação. Ao clicar no botão “<Back to Technology”, presente na Figura 4.13, o gráfico volta ao estado inicial, que contém o primeiro nível de agregação dos dados. Como qualquer uma das colunas/tecnologias do gráfico possui valores agregados é possível efetuar este aprofundamento para cada um, em separado.

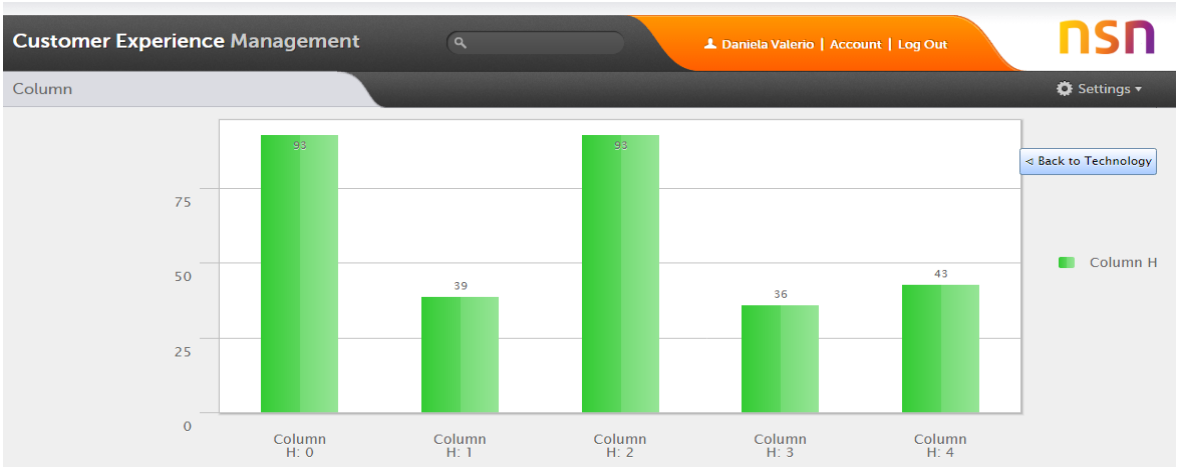

Figura 4.13 – Resultado do *drill down*

Para integrar este componente basta incluir a seguinte diretiva:

```
<nsn-columnchart-extensible filejson="input.json"></nsn-columnchart-extensible>
```

Para além destes tipos mais complexos de gráficos foi implementado um mais simples, o gráfico do tipo tarte. Este tipo de gráfico por norma não é muito usado mas traz algumas vantagens em relação aos descritos anteriormente.

A diretiva que permite este tipo de gráfico foi definida da seguinte forma:

<nsn-piechart filejson="http://localhost:8888/JSON_files/tests.json"> </nsn-piechart>

Analisando a Figura 4.14 é possível com este tipo de gráfico compreender de forma bastante fácil e quase automática qual o modelo que tem os valores mais elevados, em comparação com os restantes modelos representados no gráfico. Cada modelo do gráfico representa uma só série de valores, por enquanto que todos os restantes gráficos analisados permitem uma ou mais séries de valores.

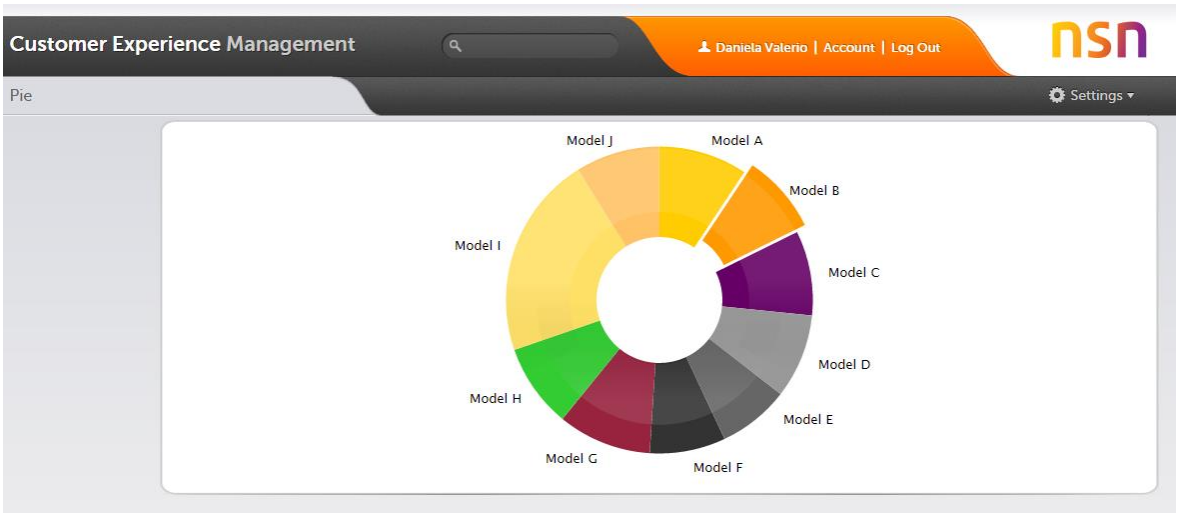

Figura 4.14 – Gráfico do tipo tarte

Os gráficos mais comuns são os gráficos de barras e colunas que no caso deste projeto podem ter dois tipos de vistas. Podem ser gráficos com as séries normais ou com as séries empilhadas. No caso da Figura 4.15 trata-se de um gráfico de barras empilhadas e no caso da Figura 4.16 um gráfico de barras normal. Esta funcionalidade foi implementada para, como se pode observar em ambas as figuras, Figura 4.15 e Figura 4.16 facilitar a visualização e compreensão dos valores num gráfico quando este tiver muitas séries para apresentar.

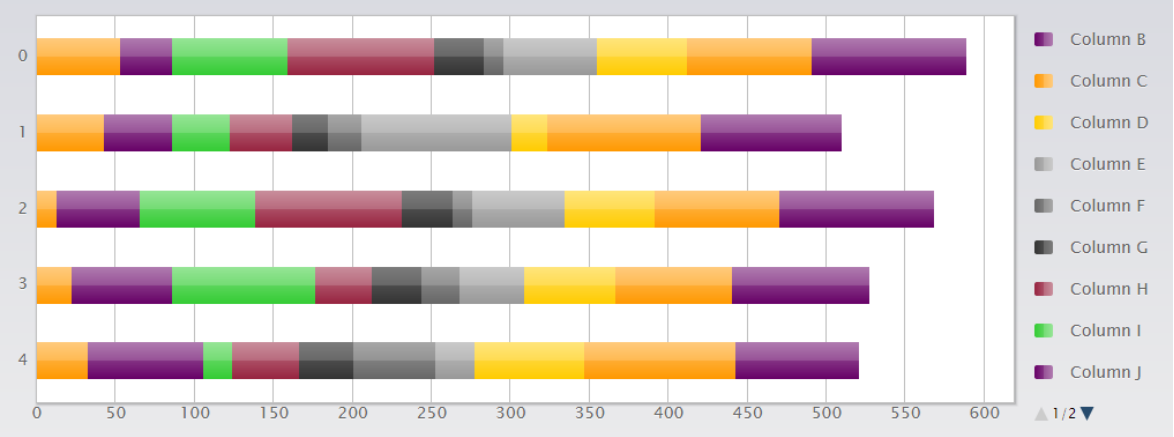

Figura 4.15 – Gráfico de barras empilhadas

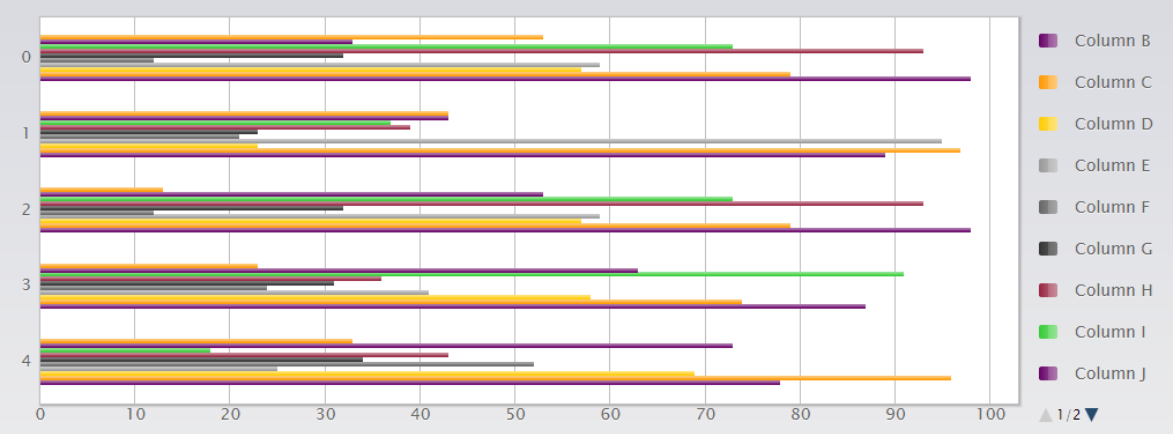

Figura 4.16 – Gráfico de barras normal

Igual funcionalidade mas neste caso nos gráficos de colunas é possível observar na Figura 4.17 e na Figura 4.18

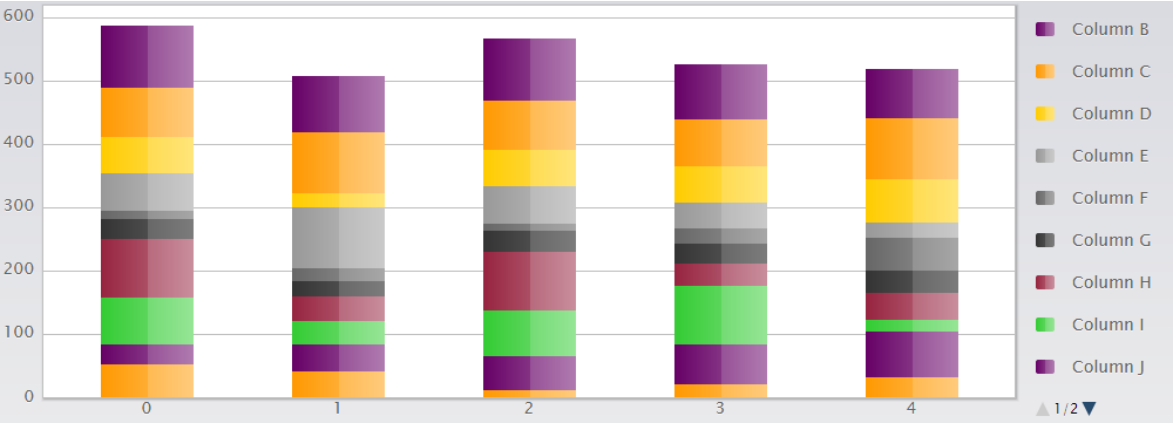

Figura 4.17 – Gráfico de colunas empilhadas

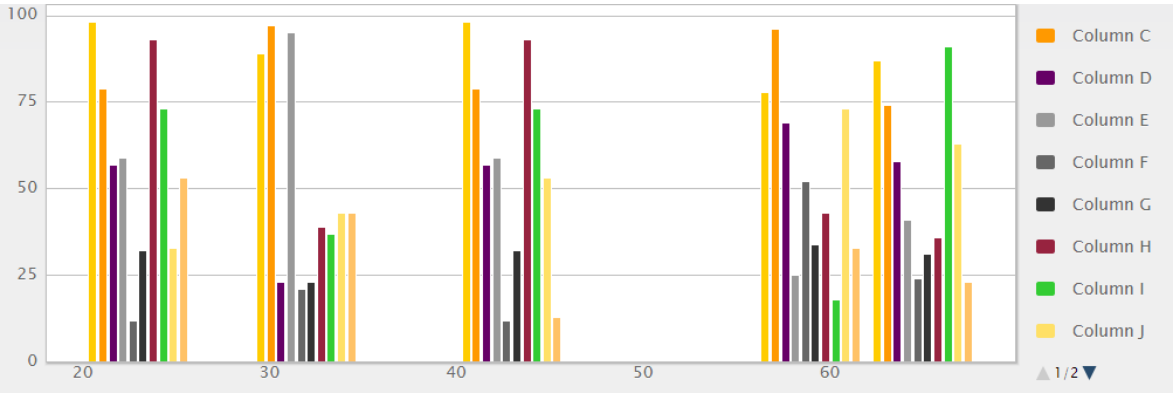

Figura 4.18 – Gráfico de barras normal

Para implementar esta funcionalidade foi necessário definir uma variável com "*data-bindig*" pois só assim é possível perceber quando é que o utilizador pretende alterar a vista normal para empilhado e vice-versa. Esta variável quando alterada é automaticamente visível a sua alteração no lado do modelo da aplicação e é redesenhado o gráfico. Esta variável faz parte das variáveis definidas e passadas como argumento de entrada na função responsável por criar os gráficos.

Para simplificação de quem pretender posteriormente utilizar as diretivas implementadas no projeto, teve-se em conta a importância de definir cada diretiva com a mesma lógica e estrutura e por isso é muito intuitivo perceber como poderá ser as diretivas para os gráficos de colunas e gráficos de barras.

- Diretiva para gráfico de barras:

```
<nsn-barchart filejson="input.json"></nsn-barchart>
```

- Diretiva para gráfico de colunas:

```
<nsn-columnchart filejson="input.json"></nsn-columnchart>
```

Os componentes web foram implementados de forma a permitir alterações de código sem que isso signifique *refactoring* do que já está implementado. O código do projeto permite um grande nível de independência entre os módulos para que alterações de código num determinado módulo, ou componente, não impliquem alterações de código noutras zonas do projeto.

Foram implementadas todas as diretivas necessárias para cada um dos componentes web. Sem dúvida que depois de implementadas as nossas próprias diretivas, a implementação de um produto com os diferentes tipos de gráficos e tabelas é muito simplificada, bastando para isso uma única linha de código por componente pretendido. Para além de que toda a complexidade e lógica JS de cada componente é completamente encapsulada do programador que utiliza estes mesmos componentes. A facilidade com que é possível introduzir ou eliminar estes componentes numa vista é impressionante, conseguindo-se assim diminuir drasticamente o tempo e esforço para

desenvolvimento de novos produtos da empresa Nokia Networks que precisem destes componentes, uma vez que se evita, repetidamente, ter de implementar toda a lógica que cada componente na realidade agrega.

4.3 Padrões de *Design* UI

Com o passar do tempo são notados determinados comportamentos recorrentes e com base nisso, definidas algumas diretivas e soluções UI. Surge assim o conceito de padrões UI, que visam resolver problemas comuns e muito recorrentes de *design*. Estes padrões são somente pontos de referência para o programador e não regras obrigatórias a implementar. No âmbito do projeto, a manipulação de informação é a questão central e como um dos objetivos futuros do projeto é visar os dispositivos móveis é muito importante, nesta fase do projeto, tentar definir vistas que possam executar tanto em dispositivos móveis, como não móveis. Neste sentido, uma das muitas preocupações a ter é a implementação de *media queries*, conceito este analisado já de seguida.

Media Queries

As *media queries* são a funcionalidade mais importante das aplicações consideradas web *responsive*. Implementando *media queries* é possível o programador construir vários *layouts* usando os mesmos documentos HTML, ou seja, implementar diferentes vistas para diferentes contextos. Para além da sua simplicidade de utilização, permitem a reutilização de código, evitando duplicação de documentos HTML por cada vista a implementar. Esta funcionalidade é muito importante tendo em conta que o principal objetivo de uma aplicação é que esta esteja funcional no maior número possível de dispositivos eletrónicos, com o mínimo de esforço. Através das *medias queries* é possível detetar algumas características base do navegador do utilizador e agir de acordo com elas. As caraterísticas mais comuns são a deteção do tamanho da janela da página web, parâmetros de orientação (vertical ou horizontal), resolução do ecrã, cores do ecrã suportadas, entre outros.

A ferramenta OT2, responsável pelo *look and feel* da aplicação, não tem implementadas quaisquer *media queries*. Mas como um dos objetivos futuros da aplicação passa pela vertente dos dispositivos móveis, tanto *tablets* como *smartphones*, esta foi uma preocupação tida em conta no projeto. Portanto, foram implementadas *media queries* para deteção de largura de ecrã. Esta funcionalidade permite o redimensionamento automático dos elementos a apresentar na página.

As *media queries* implementadas seguem as seguintes regras:

- No caso de o tamanho de ecrã detetado for igual ou superior a 600px é apresentada a vista normal, Figura 4.19, que pretende atingir dispositivos com ecrãs de grandes dimensões.

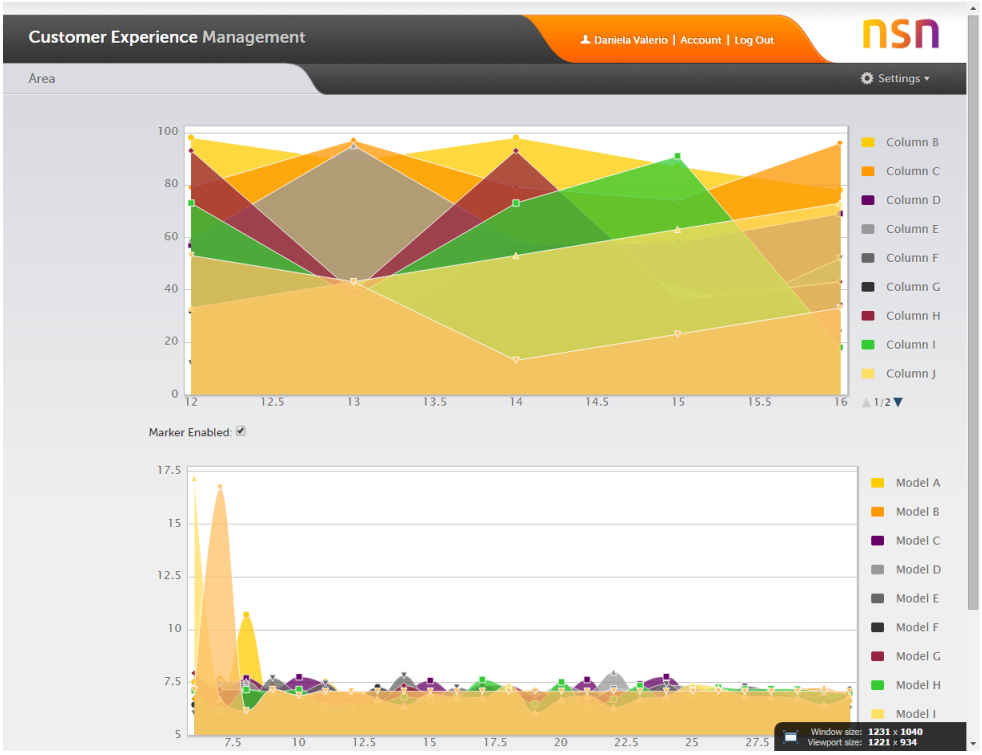

Figura 4.19 – Resolução superior a 599px de largura

- Caso contrário, significa que o tamanho de ecrã é igual ou inferior a 599px e por isso, o tamanho dos componentes é diminuído para 50% do seu valor original. O resultado tem a vista apresentada na Figura 4.20, que pretende ser a vista para os dispositivos móveis, conhecidos por terem ecrãs mais pequenos.

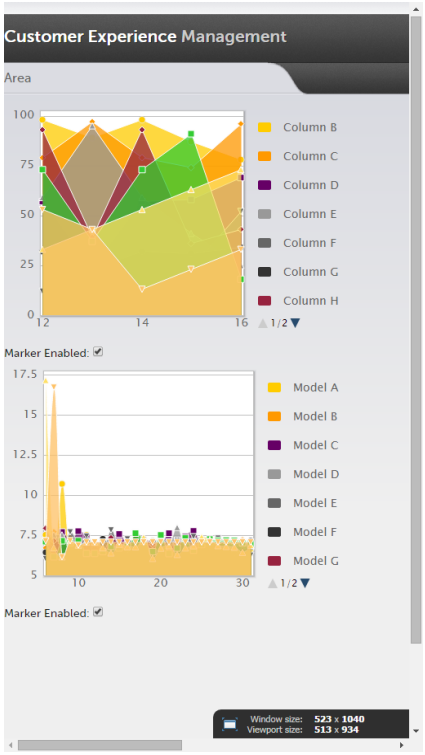

Figura 4.20 – Resolução inferior a 600px de largura

Este tipo de regras é muito importante principalmente para aplicações que sejam ou se prevê que venham a ser executadas por dispositivos com ecrãs mais reduzidos.

4.4 Test-Driven Development

Test-driven Development é uma metodologia muito falada nos últimos tempos mas ainda pouco usada. Esta metodologia deve acompanhar todo o processo de desenvolvimento de software.

Os testes têm maior importância do que aquela que por norma lhe é atribuída. A implementação de testes serve como garantia de maior longevidade na qualidade e possível reutilização do código e são a única garantia de que posteriores implementações de código não criem conflito com os objetivos e requisitos anteriormente impostos para a aplicação. São os testes que permitem ao programador efetuar *refactoring* do código sem que este tenha medo de estragar a lógica já implementada, pois caso alguma condição pelo novo código produzido não cumpra os requisitos, os testes irão notificar de imediato o programador. Esta é uma questão muito importante para código considerado *legacy code* e não podemos negar que todo o código passa eventualmente a ser *legacy code*, é uma questão de tempo inevitável e inerente a grandes projetos com grande longevidade.

4.4.1 Testes de Unidade

Tendo em conta a importância de testes num projeto, foram implementados testes de unidade relativos aos requisitos esperados da aplicação.

Por exemplo, foram implementados testes que verificam que determinada função recebendo determinados parâmetros de entrada irá reproduzir o resultado esperado. Caso, esta condição não se verifique o teste falha e o programador sabe que alguma coisa na implementação daquela função não está funcional.

Um exemplo de um teste unitário implementado no projeto é:

```
it("should have a class nsnTable on a div", function () {
 var div = element.find('div');
 expect(div).toHaveClass('nsnTable');
});
```

Este teste muito simples, como a própria descrição explica, verifica se está definida uma classe de CSS com nome 'nsnTable' num elemento HTML 'div'.

Quando os testes de unidade são executados e caso nem todos passem, o IDE Webstorm apresenta as janelas de resultados, como representa a Figura 4.21 e Figura 4.22.

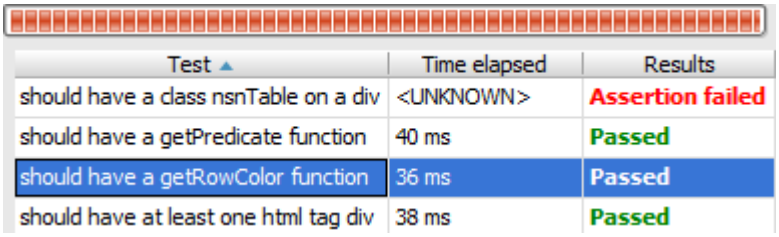

Figura 4.21 – Relatório de testes com falhas

Todos os testes que falham têm uma descrição com a razão da falha. Basta analisar a descrição e ver o que era esperado em comparação com o resultado obtido da execução do teste. A Figura 4.22 tem um exemplo de uma descrição com o erro do teste que falhou.

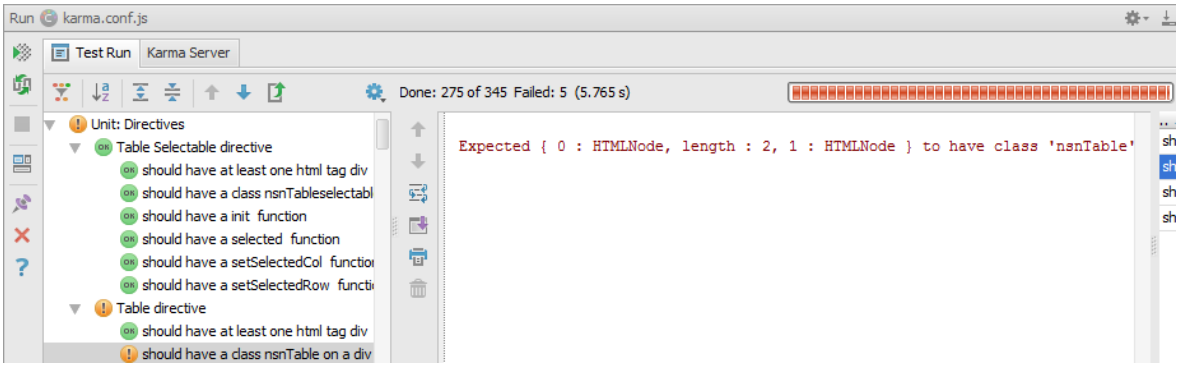

Figura 4.22 – Exemplo de um teste falhado

Existe a possibilidade de indicar quais os testes que pretendemos que sejam ignorados e que por isso não serão executados pela ferramenta de testes. Para isso e considerando que se está a utilizar a ferramenta Jasmine, basta usar um "x" antes do 'it' nesses testes [11]. Um exemplo para que um teste passe a ser ignorado:

```
xit("should have a class nsnTable on a div", function () {
 var div = element.find('div');
 expect(div).toHaveClass('nsnTable');
});
```

Desta forma, alguns testes seriam ignorados e essa informação é visível no relatório final de execução dos testes.

Analisando a Figura 4.23:

- ✘ 5 testes falharam (**F**);
- 65 testes foram ignorados (**I**);
- ✓ 283 testes passaram (**P**).

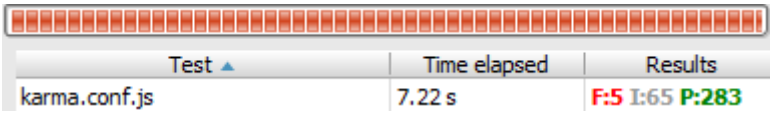

Figura 4.23 – Relatório final de testes com falhas e testes ignorados

De notar que também é apresentado o tempo de execução total dos testes, como se pode verificar na Figura 4.23 na coluna "Time elapsed".

Em contrapartida com o exemplo de execução de testes da Figura 4.22 em que existiram testes que falharam, existe o exemplo de execução de testes da Figura 4.24, em que todos os testes executados tiveram sucesso e não existe nenhum teste a ser ignorado.

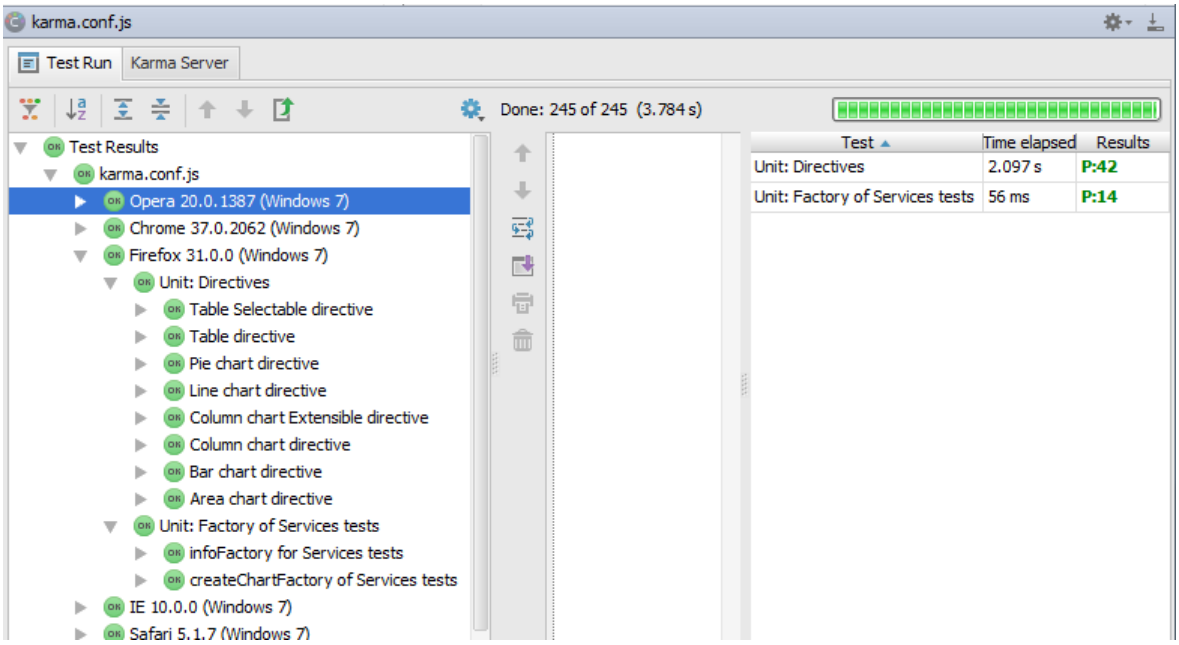

Figura 4.24 – Execução com sucesso dos testes de unidade

Para testar partes de código mais complexas foi necessário recorrer a *mocks* de forma a injetar código pretendido para somente testar a parte que interessava, pois na implementação real, existem muitas funcionalidades que só conseguem ser testadas caso determinados módulos e outros acontecimentos sejam de alguma forma simulados. A biblioteca Jasmine permite o uso de módulos de *mocks* e espiões que permitem implementar testes de unidade de maior complexidade e por sua vez aumentar o nível de cobertura dos testes no código total do projeto.

Foram implementados testes de unidade para garantir que o resultado implementado era o resultado esperado.

Uma boa norma seria executar estes testes no máximo possível de navegadores diferentes, para garantir que os testes passam em qualquer um dos navegadores que devem ser suportados por serem requisito do projeto. Mas na realidade, executar todos os testes para todos os navegadores, corresponde a um gasto de recursos demasiado dispendioso, não sendo possível na realidade de sustentar. Neste caso, os testes de unidade foram executados usado os cinco navegadores web mais comuns:

- Firefox 30.0.0,
- Chrome 36.0.1985,
- Safari 5.1.7,
- Opera 20.0.1387
- IE 10.0.0

Todos os navegadores são para o sistema operativo Windows 7. O *plugin* Karma permite facilmente configurar o projeto para que tal seja possível, basta acrescentar no ficheiro de configuração do Karma as *tags*: 'Chrome', 'Firefox', 'Opera', 'Safari', 'IE'. Desta forma será lançado um servidor por cada navegador, como é possível analisar na Figura 4.25 e o número total de testes passa a ser multiplicado pelo número de navegadores que se pretende testar.

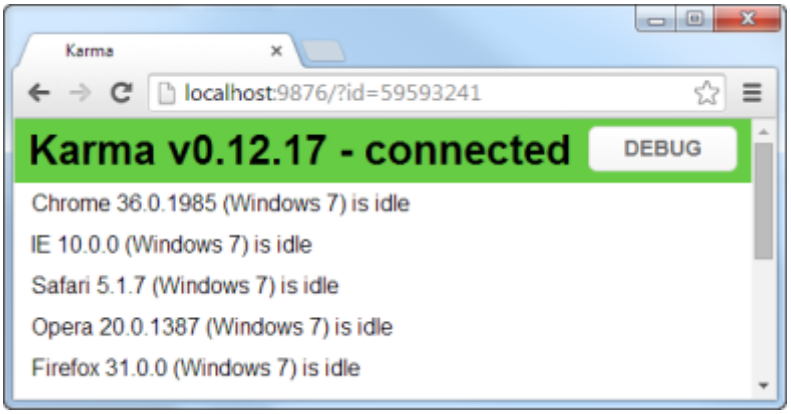

Figura 4.25 – Servidores em execução para o Karma

4.4.2 Testes *End-to-End*

Vamos agora analisar os testes *end-to-end*. Este tipo de testes pretendem testar a aplicação como se se trata-se de um utilizador real da aplicação.

Neste caso e por ser um projeto implementado com a ferramenta AngularJS, para a implementação destes testes tinha de ser utilizada a biblioteca Jasmine, por ser a única interpretada pela ferramenta Protractor, a qual executa estes testes nos diferentes *webDrivers* instalados.

Os testes *end-to-end* implementados no projeto são bastante simples, como por exemplo, detetar que quando é clicado no menu “Settings” e posteriormente escolhida a opção “Line”, que o título corresponde a “Line” e que é criado um gráfico de linhas como seria de esperar. Esta lógica é implementada para as diferentes opções da lista do menu “Settings”, que na totalidade são 8, o que implica a implementação de 8 testes diferentes.

Outro tipo de teste implementado corresponde à vista que é esperada por *default*, quando a aplicação é executada e não foram efetuados clicks ou modificações na vista, espera-se que exista o componente “Data View” que tem definido o `id = “dataView”.` Considerando que a aplicação executa localmente, no `url = “http://localhost:8888/”,` o teste implementado verifica se realmente existe este id definido depois de executada a página inicial.

```
(function () {
  "use strict";
  describe('E2E Tests: Charts, function() {

    var ptor;
    beforeEach(function() {
      ptor = protractor.getInstance();
    });

    it('should load have a dataView id, function() {
      ptor.get('http://localhost:8888/');
     expect(ptor.findElement(protractor.By.id('dataView')).getText()).toBeDefined(true);
    });
  });
}());
```

De notar que antes da implementação de qualquer teste é necessário primeiro obter a instância da ferramenta Protractor [12]. Para tal é utilizado o método "beforeEach" da biblioteca Jasmine.

Para executar estes testes foi configurada uma tarefa no ficheiro de configurações do Grunt. Desta forma, introduzindo o comando: "grunt test:e2e" todos os testes deste tipo são executados. Quando todos os testes forem executados é apresentado, também através da consola, o resultado dos testes: quantos testes falharam e quantos testes passaram com sucesso. O relatório final é apresentado na Figura 4.26.

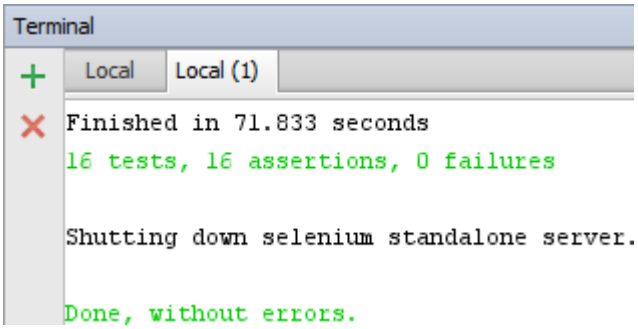

Figura 4.26 – Relatório final da execução dos testes *end-to-end*

Quando nem todos os testes passam, o relatório final da execução dos testes passa a vermelho. Igualmente com a indicação de quantos testes passaram e quantos falharam.

Para que múltiplos *webDrivers* executem ao mesmo tempo estes testes para os diferentes navegadores web foi configurado no ficheiro "protractor-conf.js" a seguinte propriedade:

```
multiCapabilities: [
        {'browserName': 'firefox'},
        {'browserName': 'chrome'},
        {'browserName': 'internet explorer'},
        {'browserName': 'opera'},
        {'browserName': 'safari'}]
```

Executar os testes depois destas configurações implica que cada teste será executado por cada um destes navegadores. Por causa dos testes estarem a executar todos na mesma máquina e por não existirem recursos suficientes para que todos os *webDrivers* executem normalmente, alguns testes podem falhar por questões de *time-out*, tal como aconteceu no projeto. O navegador Google Chome foi o único navegador em que falharam testes. O relatório final destes testes é apresentado na Figura 4.27.

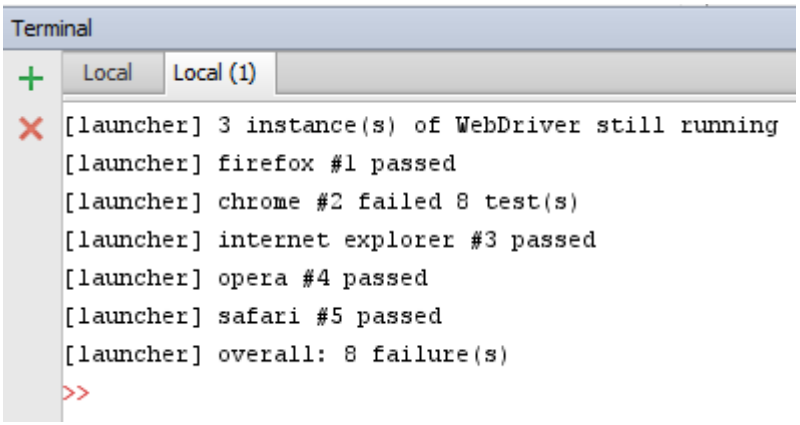

Figura 4.27 – Execução dos testes em vários navegadores em concorrência

Em conclusão, a implementação deste tipo de testes é muito importante mas para a execução de vários *webDrivers* para testar a aplicação é fortemente recomendado que estes sejam executados em diferentes máquinas e com muitos recursos disponíveis. Somente desta forma se consegue garantir que quando um teste falha é por erro da funcionalidade da aplicação e que por isso tem de ser corrigido e não por questões de *time-out*.

4.5 Ocean Touch

Nos últimos meses, a empresa Nokia Solutions and Network (NSN) passou a ser propriedade da Nokia Networks e como tal o *look and feel* das aplicações têm de acompanhar esta mudança. Neste sentido, surgem novas regras para as folhas de estilo, imagens e fontes referentes ao novo tema baseado em tons de azul, branco e cinza. Esta mudança implica atualizar o *front-end* de todos os componentes web criados no projeto de forma a corresponder a este requisito. Para tal é substituído o Orange Touch 2 (OT2) antes utilizado, pelo Ocean Touch (OT3). No caso do projeto, decidiu-se aproveitar o trabalho feito relativamente ao OT2 e possibilitar o utilizador de escolher qual o tema que mais lhe agrada.

Na Figura 4.28 é apresentado o resultado da aplicação utilizando a biblioteca OT3.

Figura 4.28 – Ocean Touch

As cores dos gráficos e *degradés* também são afetados, bem como o tipo e tamanho de letra. A nível de funcionalidades estas são esperadas terem o mesmo comportamento que as do Orange Touch 2. O projeto tendo um nível de independência elevado entre os componentes web implementados e o *look and feel*, a alteração do tema é relativamente fácil e rápida de implementar. Esta capacidade de alteração dos formatos e regras de estilo é muito importante, porque, como aconteceu no decorrer da implementação do projeto, o *look&feel* de um produto vai sofrer alterações ao longo do tempo, sendo imprescindível que estas alterações sejam rápidas e fáceis de implementar.

4.6 Tecnologia *Mobile*

Existindo cada vez mais utilizadores e clientes Nokia interessados em aplicações móveis, seria certamente uma mais-valia para o projeto usufruir da capacidade da ferramenta PhoneGap[27]. O PhoneGap permite converter uma aplicação web implementada em HTML5, CSS e JS e torná-la

[27] http://phonegap.com/

compatível para versões móveis. Desta forma, criando uma única aplicação é possível obter outra capaz de executar em dispositivos móveis.

Portanto, utilizando as capacidades da ferramenta PhoneGap foi possível implementar uma aplicação *mobile*, usando o mesmo código HTML5, CSS e JavaScript desenvolvido para a aplicação *desktop*. Desta forma foi possível observar o nível de desempenho da aplicação neste tipo de dispositivos, bem como de eventuais alterações que têm de ser tidas em conta por se tratar de uma aplicação de *touch* e não de *clicks.*

Como ponto de partida foi implementada uma aplicação *mobile* que executa somente em sistemas operativos android mas é importante salientar que o PhoneGap possibilita a criação para outros sistemas operativos mobile. Como dispositivo *mobile* de execução, foi utilizado um telemóvel Samsung Galaxy Ace 2, que tem a versão android 4.1.2. De seguida são apresentados alguns *sreenshots* da aplicação.

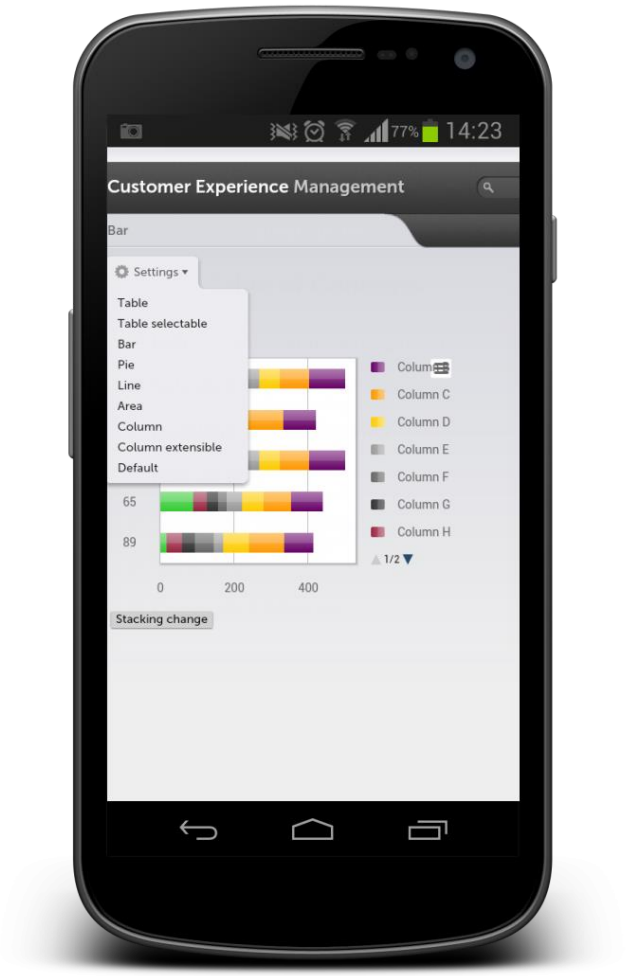

Figura 4.29 - Diferentes tipos de gráficos

Figura 4.30 – Exemplo vista gráfico de linhas

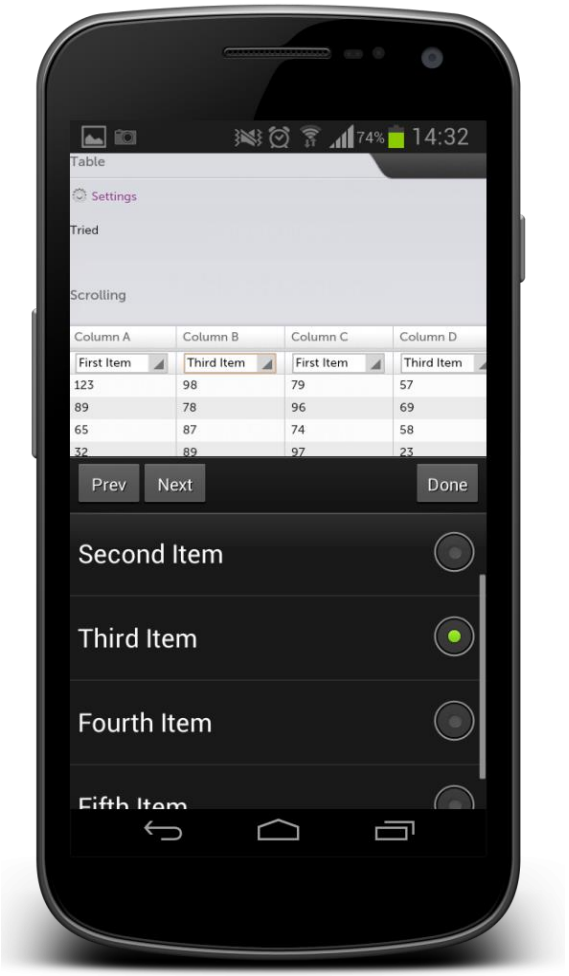

Figura 4.31 – Exemplo de filtros

Na Figura 4.29 apresentado um menu de seleção dos diferentes tipos e vistas que podem ser apresentados. Por detrás do menu está visível o tipo de gráfico de colunas empilhado. De notar, que como o número de elementos na legenda do gráfico é elevado para o tamanho do gráfico, a legenda tem ícones que permitem fazer o *scroll* da legenda.

Na Figura 4.30 é apresentado uma vista do componente de gráfico de linhas com possibilidade de ocultar os marcadores. Desta forma é possível analisar o tempo de resposta da aplicação para interações do utilizador com a aplicação.

No caso da Figura 4.31 é possível analisar uma capacidade automática do HTML. Como é detetado que a aplicação está a executar num navegador móvel e foi selecionado na tabela um elemento HTML com uma lista de opções é lançado automaticamente, no dispositivo móvel, uma lista desses menus, com botões de “Prev”, “Next” e “Done”.

No caso da opção de rotação do dispositivo estar ativa e o utilizador mude para a vista horizontal, o tamanho de ecrã detetado, no dispositivo onde foi testada a aplicação, passa a ser superior a 600px e por este motivo a vista é redimensionada, como seria de esperar.

A aplicação *mobile* tem certamente de sofrer muitas alterações para que fique realmente funcional neste tipo de tecnologia mobile. De qualquer forma, esta implementação também permitiu avaliar se o uso da ferramenta PhoneGap seria viável no futuro e qual o tempo necessário para converter toda a aplicação numa aplicação *mobile*. Com as facilidades que a ferramenta fornece, sem dúvida que o tempo e esforço são muito reduzidos e por isso é sem dúvida uma mais valia usufruir desta capacidade, em detrimento da necessidade de implementar de raiz uma aplicação *mobile*.

4.7 Síntese

Neste capítulo procurou-se dar exemplos reais de como utilizar os componentes web e de quais as diretivas implementadas. Foram analisadas as funcionalidades que são comuns a todas as diretivas, bem como outras que só fazem sentido em determinado tipo de componente, como por exemplo a aplicação de filtros e ordenação de dados nas tabelas. Toda a lógica comportamental implementada nas diretivas permite aumentar o nível de interação entre utilizador e componentes e vice-versa e é esta interatividade que cativa o utilizador.

A implementação de diretivas do AngularJS é sem dúvida uma mais-valia na implementação de aplicações web com código realmente reutilizável e escalável. Implementando as diretivas e os serviços por elas necessários foi possível implementar com sucesso o objetivo principal, a criação de componentes web HTML5 reutilizáveis.

A importância da implementação de testes, tanto de unidade como *end-to-end* é agora realmente compreendida, pois permite deteção de erros atempadamente, bem antes que o produto seja entregue ao cliente. Num ambiente empresarial com a dimensão da Nokia Networks a correção de erros atempadamente, significa, evitar custos significativos.

5 Conclusões

Ao longo da implementação do projeto foram feitas demonstrações para diferentes audiências, desde pessoas que pertencem à empresa Nokia em Aveiro, bem como para outros elementos da Nokia Networks que desenvolvem outros produtos da empresa mas que sofrem da mesma necessidade de implementação de componentes web. Falamos de diferentes pontos do globo, tanto para elementos da Nokia Networks em Lisboa, bem como para a Finlândia e Índia. Estas demonstrações permitiram receber *feedback* e trocar ideias sobre a arquitetura do projeto, bem como da abordagem e das ferramentas escolhidas. Concluiu-se que a abordagem adotada para a implementação dos componentes web foi bem aceite e muito apoiada, o que significa que foram feitas boas escolhas.

Com base nos objetivos e requisitos iniciais do projeto de dissertação e por terem sido todos correspondidos, a implementação do projeto é considerada um sucesso. A proposta de dissertação serviu realmente como rampa de lançamento para todo o processo de migração do NPM e neste momento mais componentes web estão a ser implementados usando as mesmas ferramentas e conceitos analisados e adquiridos ao longo de todo este processo de aprendizagem e implementação.

Apesar das linguagens HTML5, CSS e JS não serem, ao início da implementação do projeto, linguagens com as quais tivesse uma vasta experiência, neste momento, abriram-me as portas para novas oportunidades e para a implementação de aplicações *front-end*. É com orgulho que posso dizer eu adquiri muitos conhecimentos e implementei um projeto com base em ferramentas e tecnologias web muito inovadoras, das quais ainda poucos programadores tiveram oportunidade e motivação de explorar.

5.1 Trabalho futuro

5.1.1 Documentação do Projeto

Todas as aplicações da empresa Nokia têm documentação de ajuda para o utilizador. Esta documentação é feita por pessoas especializadas nesta área e que pertencem à empresa. Este tipo de documentação é importante para ajudar o utilizador a usar a aplicação e a conhecer as funcionalidades que a mesma permite.

Neste tipo de projetos que se prevê que sejam de grandes dimensões e usados por vários programadores outro tipo de documentação importante é a documentação do próprio código, bem

como de ficheiros README para ajuda no processo de instalação e construção de todo o ambiente de desenvolvimento da aplicação e outras questões referentes a comandos que se possam usar para lançamento de testes ou outras tarefas importantes no desenvolvimento da aplicação. Já foi de momento implementada alguma documentação a este nível, com os requisitos e comandos suportados pela aplicação mas mesmo assim, este tipo de documentação está em constante mudança. Cada vez que por exemplo são introduzidos novos *plug-ins* suportados pela aplicação e acrescentadas tarefas no Grunt é necessário avisar como é possível lançar essa tarefa.

5.1.2 Escalabilidade e Extensibilidade do Projeto

Sendo o objetivo deste projeto integrar futuramente estes componentes em projetos de grande dimensão, inicialmente na aplicação NPM da Nokia e futuramente noutras com requisitos equivalentes, este será sem dúvida um trabalho futuro de grande importância. Será necessário substituir o código agora existente em Flex pelos módulos HTML5 implementados, iniciando assim por partes a migração do projeto NPM de Flex para HTML5.

Para concluir todo o processo de migração do código, terão de ser implementadas todas as funcionalidades que atualmente o projeto NSN comporta em Flex, bem como de novas funcionalidades. De notar que no futuro, estes componentes terão de ser integrados e utilizados no projeto NPM mas também se pretende que sejam integrados e utilizados por outros projetos da empresa.

Referências

[1] S. Mavrody, "Sergey's HTML5 & CSS3 Quick Reference. 2nd Edition", Belisso Corp., 2012.

[2] W3Techs, “Usage Statistics of Client-side Programming Languages for Websites”, Online: http://w3techs.com/technologies/overview/client_side_language/all. [Acedido em Outubro 2014].

[3] D. Crockford, “JavaScript: The Good Parts”, O'Reilly, 2008.

[4] W3Techs, “Usage Statistics and Market Share of JavaScript Libraries for Websites”, Online: http://w3techs.com/technologies/overview/javascript_library/all. [Acedido em Setembro 2014].

[5] A. Lerner, “ng-book, The Complete Book on AngularJS”, FULLSTACK.io, 2013.

[6] GitHub, “Projects using AngularJS”, Online: https://github.com/angular/angular.js/wiki/Projects-using-AngularJS. [Acedido em Novembro 2013].

[7] GitHub, “Backbone Project”, Online: https://github.com/jashkenas/backbone/. [Acedido em Novembro 2013].

[8] GitHub, “Projects and Companies using Backbone”, Online: https://github.com/jashkenas/backbone/wiki/Projects-and-Companies-using-Backbone. [Acedido em Setembro 2014]

[9] B. G. &. S. Seshadri, “AngularJS,” O'Reilly, 2013

[10] Igor Minar - Google, “AngularJS,” Google, Online: https://plus.google.com/+AngularJS/posts/aZNVhj355G2. [Acedido em Setembro 2014].

[11] E. Hahm, “JavaScript Testing with Jasmine,” O'Reilly, 2013.

[12] Google, “AngularJS: Developer Guide: E2E Testing,” Online: https://docs.angularjs.org/guide/e2e-testing. [Acedido em Setembro 2014].